美河图书
http://book.eimhe.com

专业图书 it 图书 0day 图书 期刊杂志下载基地

本站提供的书籍均为网上搜集,所有权归属原作者。只可学习试看，请于下载 24 小时内删除。切勿用于商业用途，否则由此引起一切后果自负。

如果该书籍侵害到您的版权请立即写信通知我们，我们将在24 小时之内删除。

默认解压密码：book.eimhe.com

推荐使用：WinRAR V3.3 以上版本解压本站书籍

美河学习在线 http://www.eimhe.com

最新、最全的视频资料下载基地

HIFI 音乐地带： http://music.eimhe.com

音乐软硬件 流行 古典 纯音乐 MV 每天更新

Java开发利器

附：视频讲解
本书源代码

强锋科技
陈刚 编著

Eclipse

从入门到精通

- 国内第一本详细讲解SWT/JFace开发的书
- 国内第一本详细讲解如何开发Eclipse插件的书
- 一本写给程序员的书，是作者实际项目经验的总结
- 内容精简，去除了大量的冗余代码，含金量高
- 配多媒体光盘，详细讲解软件的安装和配置

清华大学出版社

Java开发利器

Eclipse
从入门到精通

丛书特色：

- ✔ 作者全都是有丰富编程经验的一线开发人员
- ✔ 全面攻克Java开发领域的前沿技术
- ✔ 极大地提升您的Java应用开发水准
- ✔ 确保技术的实用性和深入性
- ✔ 贯穿丰富的应用实例，真正做到学以致用
- ✔ 配多媒体光盘讲述界面操作，既节省篇幅，又容易上手

光盘内容：

- ✔ 全程录制本书涉及软件的基本操作
- ✔ 本书所涉及的源代码

ISBN 7-302-11135-9
9 787302 111351 >
定价：48.00元（附1张光盘）

Java 开发利器

Eclipse 从入门到精通

强锋科技

陈　刚　编著

清华大学出版社

北　京

内 容 简 介

本书共分为四篇。“起步篇”介绍了 Eclipse 及相关插件的安装，还介绍了一些 Eclipse 开发环境的基本使用技巧；“SWT/JFace 篇”详细介绍了 SWT、JFace 的使用；“插件开发篇”介绍了 Eclipse 插件开发的各个步骤，并给出了一个基于数据库开发和面向对象分析设计的完整插件实例；“Web 开发篇”以 Tomcat+Lomboz + Struts + Hibernate 为开发环境，详述了其下载、安装、配置和开发的全过程。

本书以 SWT/JFace 和 Eclipse 插件开发为核心内容，并兼顾了 Web 开发，而 Eclipse 开发环境的使用只占极少部分。

本书内容循序渐进，实例丰富，操作及开发步骤讲解详细，配书光盘含有各主要章节的操作视频演示，是 Eclipse 开发者不可多得的教程。

版权所有，翻印必究。举报电话：010-62782989　13501256678　13801310933
本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
本书防伪标签采用特殊防伪技术，用户可通过在图案表面涂抹清水，图案消失，水干后图案复现；或将表面膜揭下，放在白纸上用彩笔涂抹，图案在白纸上再现的方法识别真伪。

图书在版编目（CIP）数据

Eclipse 从入门到精通/强锋科技编著；陈刚/编著. —北京：清华大学出版社，2005.6
（Java 开发利器）
ISBN 7-302-11135-9

Ⅰ. E…　Ⅱ. ① 强…　② 陈…　Ⅲ. JAVA 语言-程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2005）第 054796 号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
　　http://www.tup.com.cn　　邮　　编：100084
　　社 总 机：010-62770175　　客户服务：010-62776969
组稿编辑：欧振旭
文稿编辑：刘　丽
封面设计：姜凌娜
版式设计：冯彩茹
印 刷 者：北京国马印刷厂
装 订 者：北京市密云县京文制本装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：33　字数：762 千字
版　　次：2005 年 6 月第 1 版　2005 年 6 月第1 次印刷
书　　号：ISBN 7-302-11135-9/TP · 7359
印　　数：1～5000
定　　价：49.00 元(附 1 张光盘)

前　言

本书的形成

2003 年末，我离开深圳工作了一年的管理岗位，收拾行囊来到北京，决定做一年程序员以更新自己的技术知识结构。其后得到了 IBM.CRL 的一份工作，是用 Eclipse 插件的方式为台湾某银行开发一个项目管理软件。

去的当天，同事演示了一个让我两眼放光的 Demo，那是一个如此优秀和吸引人的 GUI 界面，让人难以相信它是用 Java 开发的。我曾经用 AWT/Swing 开发过 GUI 程序，但开发出来的界面总是感觉不好看。老实说我并不希望自己再使用 AWT/Swing 来开发软件。

Eclipse 插件开发需要用到 SWT/JFace 和其插件扩展机制的知识，当时资料很缺乏，仅在网上有一些零星的文章，但这些文章用来入门还可以，要正式开发则是远远不够的。幸运的是，我的同事都有着丰富的 Eclipse 插件开发经验，他们给了我很多的帮助。

现在，很多大公司都聚集在 Eclipse 的开源大旗之下，这里面也包括 Borland 公司。层出不穷的插件和应用不断丰富着 Eclipse 的世界。在国外，Eclipse 的开发和应用已经如火如荼，而国内在这方面还处于萌芽状态，主要原因还是缺乏相关的学习资料。我认为在国内推广 Eclipse 是一件很有意义的事，2004 年 10 月，我辞职在家专心著书，希望能把自己的开发经验集结成册，向同行们介绍 Eclipse 这种极有发展前途的开源软件。

本书面向的读者群

现在国内的 Java 程序员大致分成两类：一类是从事 C/S 模式开发的，他们主要用 AWT/Swing 来写胖客户端程序；一类是从事 B/S 模式开发的，他们主要用 J2EE 和各种轻量级框架来写 Web 应用。本书兼顾了这两类读者的需求。

第 1 篇主要是一些 Eclipse 的基础应用，对于所有 Java 程序员都通用。内容包括 Eclipse 的安装与配置、Eclipse 集成开发环境的使用技巧、CVS 的安装与配置。

第 2 篇及第 3 篇主要针对 Java 胖客户端的开发者，讲解了 SWT/JFace 中各组件的使用，并且详细介绍了开发一个完整的 Eclipse 插件所需的具体知识和开发流程。

第 4 篇主要针对 Web 开发者，详细讲解了 Tomcat、Lomboz、Struts、Hibernate 等在 Eclipse 中的安装与配置，并且用一个贯穿全篇的实例来演示如何综合运用 Struts、Hibernate 来开发 Web 应用。这部分内容要求读者对 JSP 有一定的了解，但不要求读者了解 Struts、Hibernate。

本书的内容结构

第 1 篇为“起步篇”，共有 5 章。主要内容有：Eclipse 及其多国语言包的下载、安装与配置；辅助 GUI 开发的 SWT Designer 插件的下载、安装与配置；Eclipse 集成开发环境的介绍和主要使用技巧；CVS 服务器的安装及 Eclipse 中 CVS 客户端的使用等。

第 2 篇为“SWT/JFace 篇”，共有 11 章。主要内容有：SWT 的事件模型；SWT 的各种常用组件及布局管理器的使用；图像和 SWT 多线程；JFace 的表格、树、对话框的使用等。

第 3 篇为“插件开发篇”，共有 6 章。主要内容有：插件扩展点的使用；插件的国际化；用 POI 来输出报表；项目的打包与发行。最重要的是第 22 章，给出了一个基于数据库开发和面向对象分析设计的完整插件实例。

第 4 篇为“Web 开发篇”，共有 4 章。主要内容有：如何在 Eclipse 中搭建 Web 开发环境；如何在 Eclipse 中使用 Struts、Hibernate 来开发 Web 应用。

和市场上其他书籍相比，本书有何不同

现在国内市场上已经有了几本 Eclipse 相关的书籍，这对 Eclipse 的普及很有益。但有的书不是太过于偏向 Eclipse 界面的使用，重复了 Eclipse 中文帮助中的内容，便是翻译国外的书，这些书不是很切合国内软件开发者的实际需求，而且在内容和写法上也不符合中国人的思维方式和阅读习惯。

本书从实际开发软件的需求出发来介绍 Eclipse。本书淡化了 Eclipse 开发环境的使用，强化了 SWT/JFace、插件开发和结合流行框架的 Web 开发书中的三部分内容。我本身也是一名程序员，本书的大部分内容都是在实际项目开发中的经验总结，例子也大多脱胎于实际项目，所以这本书应该更切合中国读者的需要。

本书特点和优势

（1）实例丰富，用代码进行讲解。本书的写作理念是“代码是最好的教程”，所有知识点都有详实可运行的代码，并且这些代码都附有注释、说明及运行效果图。

（2）注重实践，有具体的项目背景。本书是面向实际项目开发而写，不谈高深的理论，只讲开发时的实际应用。每一个知识点都以实际开发的需要来讲解，步骤详细，可操作性强。并且还用“建议”的方式，给出了作者开发实践中的经验总结。

（3）行文流畅，通俗易读。本书有大量的附图，并且文字尽量避免艰深生涩，讲解内容循序渐进，有迹可寻。

（4）内容超值，附多媒体视频。为了保证读者在各种安装配置中能够成功，本书还特意制作了视频操作录相。这既节省了篇幅，为读者省下了一笔开支，同时还方便了读者学习。

（5）网上答疑，随时解答读者的问题。为本书专门开通了 blog，网址是 http://http://www.chengang.com.cn，读者可以直接和作者交流，答疑解惑。另外，作者的邮箱是 glchengang@yeah.net，读者也可以通过邮箱和作者联系。

提示

本书中软件的下载地址有可能随时间而发生变化，读者可以从网站 http://glchengang.go.nease.net/ 或 http://www.chengang.com.cn 找到最新的软件下载地址。

感谢

在此，衷心感谢我的父母，没有他们的精心照顾，我是无法完成此书的；感谢张增强

先生，他在排版及内容安排方面给了我很多的好意见；感谢同事黄俊、张浩、黄钰，他们在工作中给了我很多 Eclipse 技术方面的帮助；感谢方正公司的武向辉，是他把我引荐给了张增强先生；感谢众多网友的厚爱，是他们提出了宝贵的意见，特别要感谢 Scott Woo，是他指出了我在 CVS 的一个重要错误。

由于本书内容繁杂，写作时间仓促，加之本人水平所限，书中难免存在不当之处，恳请广大读者批评指正。

陈　刚

2005 年 6 月于桂林临桂

目　录

第1篇　起步篇

第1章　Eclipse简介 2
1.1　Eclipse的历史 2
1.2　Eclipse的体系结构 2
1.3　优秀的图形API：SWT/JFace 3
1.4　开放式、可扩展的IDE 4
1.5　全中文化的帮助文件 4
1.6　Eclipse是开放源代码的 5
1.7　本章小结 5

第2章　安装Eclipse及多国语言包 6
2.1　安装JDK 6
2.2　安装Eclipse 7
2.3　Eclipse多国语言包的安装 8
2.4　本章小结 10

第3章　安装SWT Designer 11
3.1　下载 11
3.2　安装 11
3.3　注册激活 12
3.4　其他常用插件 14
3.5　本章小结 15

第4章　Eclipse的集成开发环境 16
4.1　Eclipse界面简介 16
4.2　创建Java项目并运行 17
4.3　自定义注释及代码格式化 20
4.3.1　自定义注释 20
4.3.2　代码格式化 21
4.3.3　实践建议 21
4.4　使用Eclipse的重构功能 22
4.5　任务标记 23
4.5.1　任务标记的设定 23

4.5.2　过滤任务标记 24
4.6　在编程时随意查看JDK、Eclipse源代码 25
4.6.1　查看JDK源代码 25
4.6.2　查看Eclipse的源代码 25
4.7　在代码中搜索 28
4.8　打开类型层次结构 28
4.9　调试器的使用 29
4.10　常用快捷键 30
4.11　本章小结 31

第5章　Eclipse中CVS的使用 32
5.1　CVS简介 32
5.2　CVS服务器端的安装与配置 32
5.3　CVS客户端的配置 34
5.3.1　配置Eclipse的客户端来连接上CVS服务器 34
5.3.2　将项目提交到CVS服务器上 35
5.3.3　CVS服务器上的项目导入到Eclipse中 36
5.4　文件提交与更新的方法 37
5.4.1　CVS和VSS的不同之处 37
5.4.2　文件的提交和更新 38
5.4.3　解决文件提交的冲突 39
5.4.4　如何忽略掉不想提交的文件 40
5.4.5　实践建议 41
5.5　在CVS上为软件打包一个版本 42
5.6　将项目替换成CVS上的其他版本 42
5.7　修改旧版本的BUG 43
5.8　本章小结 44

第2篇　SWT/JFace篇

第6章　SWT概述 46
6.1　SWT简介 46
6.2　SWT中的包 47
6.3　用SWT Designer写一个Hello World 47
6.3.1　使用向导建立一个SWT/JFace Java项目 48
6.3.2　导入SWT的原生库 49
6.3.3　新建一个SWT类文件 50
6.3.4　在主窗口加入一个文本框组件 51
6.3.5　运行HelloWorld.java 52

6.4 关于SWT/JFace例程的说明 53
6.5 实践建议 54
6.6 本章小结 54

第7章 SWT/JFace的事件模型 55
7.1 事件的4种写法 55
7.1.1 匿名内部类写法 55
7.1.2 命名内部类写法 55
7.1.3 外部类写法 56
7.1.4 实现监听接口的写法 56
7.1.5 总结 57
7.2 常用事件介绍 57
7.3 在事件代码中如何访问类中的变量 58
7.3.1 访问类中变量的3种方法 58
7.3.2 Java中变量的称法和说明 59
7.4 本章小结 60

第8章 SWT常用组件 61
8.1 按钮、复选框、单选框（Button类） 61
8.1.1 一个Button类的实例 61
8.1.2 Button类的式样表 62
8.1.3 让Button应用多个式样 62
8.2 组件的常用方法 62
8.3 标签（Label类） 64
8.4 文本框（Text类） 65
8.5 下拉框（Combo类） 67
8.6 列表框（List类） 70
8.7 本章小结 71

第9章 容器类 72
9.1 面板（Composite类） 72
9.2 分组框（Group类） 73
9.3 选项卡（TabFolder类和TabItem类） 73
9.4 分割窗（SashForm类） 75
9.5 带滚动条的面板（ScrolledComposite类） 77
9.6 本章小结 78

第10章 布局管理器 79
10.1 布局管理器简介 79
10.2 充满式（FillLayout类） 80

10.2.1 FillLayout的实例1 80
10.2.2 FillLayout的实例2 80
10.2.3 使用SWT.VERTICAL式样更改排列方式 81
10.3 行列式（RowLayout类） 81
10.3.1 RowLayout的基本实例 81
10.3.2 使用rowLayout.wrap控制自动折行（默认值true） 82
10.3.3 使用SWT.VERTICAL式样控制排列方式 83
10.3.4 使用rowLayout.pack控制组件大小（默认值true） 83
10.3.5 使用rowLayout.justify控制空间伸展（默认值false） 83
10.3.6 使用RowData更改按钮的外观 84
10.4 网格式（GridLayout类） 84
10.4.1 GridLayout类的基本实例 84
10.4.2 使用gridLayout.numColumns设置列数（默认值1） 85
10.4.3 使用makeColumnsEqualWidth把组件等距分开（默认值false） 86
10.4.4 使用MarginWidth、MarginHeight、HorizontalSpacing、VerticalSpacing属性 86
10.4.5 使用GridData控制复杂布局 86
10.5 堆栈式（StackLayout类） 91
10.6 表格式（FormLayout类） 93
10.6.1 使用marginWidth、marginHeight设置边距 93
10.6.2 使用FormData的构造函数 94
10.6.3 FormAttachment类的用法 94
10.7 布局的综合实例 97
10.8 本章小结 102
第11章 其他SWT组件 104
11.1 工具栏（ToolBar类、ToolItem类、ViewForm类） 104
11.2 动态工具栏（CoolBar类、CoolItem类） 106
11.3 菜单（Menu类，MenuItem类） 108
11.4 滑动条（Slider）、刻度条（Scale）、进度条（ProgressBar） 111
11.5 画布（Canvas类） 112
11.6 表格（Table类） 114
11.7 树（Tree类） 116
11.8 表格型树（TableTree类） 117
11.9 本章小结 119
第12章 图像 120
12.1 图像（Image类） 120
12.2 图像（Image类）存在的问题 121

12.3 图像描述符（ImageDescriptor类） 121
12.4 图像注册表（ImageRegistry类） 122
12.5 本章小结 123

第13章 SWT的线程 124
13.1 SWT线程简介 124
13.2 一个SWT线程的实例 125
13.3 对11.4节进度条实例的改进 130
13.4 本章小结 131

第14章 表格（TableViewer类） 132
14.1 概述 132
14.2 让数据在TableViewer中显示出来 133
14.2.1 实例的数据模型介绍 133
14.2.2 创建数据表的实体类 133
14.2.3 数据的生成 134
14.2.4 数据在表格中的显示 136
14.3 TableViewer响应鼠标事件 139
14.4 加上右键菜单（Action类、ActionGroup类、MenuManager类） 141
14.4.1 Action类、ActionGroup类、MenuManager类介绍 141
14.4.2 创建Action和ActionGroup 141
14.4.3 在主程序中使用ActionGroup、MenuManager 143
14.5 TableViewer排序（ViewerSorter类） 143
14.5.1 编写排序器 144
14.5.2 生成排序器对象 145
14.5.3 为表格的列添加监听器及事件处理代码 146
14.6 加上工具栏（ToolBarManager类） 146
14.6.1 创建按钮的Action及将Action填充进工具栏的方法 147
14.6.2 布局调整 150
14.7 创建一个带复选框的TableViewer（CheckboxTableViewer类） 151
14.7.1 修改界面程序将表格变为带复选框的式样 151
14.7.2 修改MyActionGroup类 152
14.8 单击修改表格单元格值（CellEditor类、 ICellModifier接口） 154
14.8.1 创建一个字符串数组 154
14.8.2 修改主程序的open()方法 154
14.8.3 创建MyCellModifier类 155
14.9 其他使用技巧 157
14.9.1 表格记录的过滤 157
14.9.2 控制表格的当前选择行 158

14.9.3　给表格的单元格设置背景色 158
14.9.4　加快TableItem和记录之间的查找速度 159
14.10　本章小结 159

第15章　树（TreeViewer类）和列表（ListViewer类） **160**
15.1　概述 160
15.2　前期准备：实例所用数据模型说明 160
15.2.1　建立国家实体和城市实体 160
15.2.2　建立树结点的接口类 161
15.2.3　让国家、城市、人三个实体类实现此接口 162
15.3　让数据在树中显示出来 163
15.3.1　树的数据结构的创建 163
15.3.2　标签器和内容器的使用 165
15.3.3　标签器（实现ILabelProvider接口） 165
15.3.4　内容器（实现ITreeContentProvider接口） 166
15.4　给树加上右键菜单及取得结点的值 169
15.5　树结点的展开、收缩、新增、删除、修改 171
15.6　ListViewer类 175
15.6.1　ListViewer简介 175
15.6.2　ListViewer的实例 175
15.6.3　ListViewer常用方法 176
15.7　本章小结 176

第16章　对话框 **177**
16.1　对话框（Dialog类） 177
16.1.1　对话框简介 177
16.1.2　信息提示框（MessageDialog类） 178
16.1.3　输入值对话框（InputDialog类） 179
16.1.4　自定义对话框（Dialog类） 181
16.1.5　对话框的设置与取值 185
16.1.6　带提示栏的对话框（TitleAreaDialog类） 186
16.2　向导式对话框（WizardDialog类） 187
16.2.1　向导式对话框简介 187
16.2.2　向导式对话框实例 188
16.2.3　向导式对话框使用的注意事项 193
16.3　进度条对话框（ProgressMonitorDialog类） 193
16.3.1　进度条对话框简介 193
16.3.2　进度条对话框实例 193
16.4　其他类型对话框 195

16.4.1　信息提示框（MessageBox类）……195
16.4.2　颜色选择对话框（ColorDialog类）……197
16.4.3　字体选择对话框（FontDialog类）……197
16.4.4　打印设置对话框（PrintDialog类）……198
16.4.5　目录选择对话框（DirectoryDialog类）……203
16.4.6　文件选择对话框（FileDialog类）……203
16.5　本章小结……205

第3篇　插件开发篇

第17章　Eclipse插件开发起步……208
17.1　Eclipse插件开发概述……208
17.1.1　插件开发简介……208
17.1.2　Eclipse插件开发的优势和不足……208
17.2　插件的Hello World……209
17.2.1　使用向导一步步创建HelloWorld……209
17.2.2　以空白项目为基础创建HelloWorld……211
17.3　本章小结……215

第18章　常用插件扩展点……216
18.1　加入透视图（perspectives）……216
18.1.1　准备工作……216
18.1.2　修改plugin.xml文件，设置透视图的扩展点……217
18.1.3　建立透视图类……217
18.1.4　运行插件……217
18.1.5　总结……218
18.2　在透视图中加入视图（views）……218
18.2.1　修改plugin.xml文件，设置视图的扩展点……218
18.2.2　创建视图类……219
18.2.3　修改透视图类SamplePerspective……220
18.2.4　运行插件……221
18.3　在视图之间实现事件监听……221
18.3.1　修改View1.java、View2.java……222
18.3.2　总结……223
18.4　给视图加下拉菜单和按钮……223
18.4.1　创建ActionGroup类……224
18.4.2　修改View1类……225
18.5　加入编辑器（editors）……226
18.5.1　修改plugin.xml文件，设置三个编辑器的扩展点……227

18.5.2 创建三个编辑器类 227
18.5.3 创建IEditorInput 228
18.5.4 打开编辑器 230
18.5.5 总结 231
18.6 编辑器类（EditorPart）方法使用说明 231
18.6.1 EditorPart方法的执行情况 232
18.6.2 各方法的作用及含义 232
18.6.3 一个实例 233
18.7 加入首选项（preferencePages） 235
18.7.1 修改plugin.xml文件，设置首选项的扩展点 235
18.7.2 建立首选项各结点对应的类 236
18.7.3 运行插件 239
18.7.4 总结 240
18.8 加入帮助（toc） 240
18.8.1 修改plugin.xml文件，设置三个帮助的扩展点 240
18.8.2 编写帮助目录文件toc 241
18.8.3 创建相应的帮助文档 242
18.8.4 总结 243
18.9 弹出信息式的帮助（contexts） 243
18.9.1 修改plugin.xml文件，设置弹出信息的扩展点 243
18.9.2 编写弹出信息的设置文件HelpContexts.xml 244
18.9.3 创建弹出信息对应的帮助文件 244
18.9.4 在界面组件中设置弹出信息 244
18.9.5 运行插件 245
18.9.6 总结 245
18.10 本章小结 246

第19章 Eclipse插件的国际化 247
19.1 国际化简介 247
19.2 为国际化创建一个插件的“段项目” 247
19.3 类程序的国际化 248
19.3.1 国际化前的准备工作 249
19.3.2 创建Message类 250
19.3.3 创建资源文件（*.properties） 250
19.3.4 修改myplugin2项目的程序 252
19.3.5 运行插件，测试国际化 253
19.3.6 实践建议 253
19.4 plugin.xml的国际化 254
19.4.1 创建plugin.xml的资源文件 254

19.4.2　修改plugin.xml文件 255
19.4.3　运行插件，测试plugin.xml的国际化 255
19.5　其他XML文件的国际化 256
19.5.1　在myplugin2.language项目创建国际化所需的目录结构 256
19.5.2　修改各语言的XML文档 256
19.5.3　运行插件，测试XML文件国际化 256
19.6　使用“外部化字符串”向导 256
19.6.1　运用“外部化字符串”向导的步骤 257
19.6.2　向导完成后的处理 258
19.6.3　总结 258
19.7　本章小结 258
第20章　报表：用POI与Excel交互 260
20.1　POI概述 260
20.1.1　POI简介 260
20.1.2　POI的下载与安装 260
20.2　将数据导出成Excel的实例 262
20.2.1　创建一个空白的Excel文件 262
20.2.2　往Excel单元格中写入信息 263
20.2.3　中文化的问题 264
20.3　使用式样 264
20.3.1　日期式样及文字对齐式样 264
20.3.2　边框式样 266
20.3.3　背景色及底纹式样 267
20.3.4　合并单元格 267
20.3.5　字体式样 268
20.4　更多的用法 269
20.4.1　设置页眉页脚 269
20.4.2　冻结和分割窗 269
20.4.3　浮动文字框及在表中画图 270
20.4.4　设置打印的范围 271
20.4.5　读取及修改Excel 271
20.5　本章小结 272
第21章　项目的打包与发行 273
21.1　应用程序项目的打包与发行 273
21.1.1　简介 273
21.1.2　打包的具体操作步骤 273
21.1.3　其他得到JAR包的方式 277

21.1.4 使用第三方插件对项目打包 278
21.1.5 让用户电脑不必安装JRE环境 280
21.1.6 更进一步的完善 280
21.1.7 打包的其他说明 283
21.2 插件项目的打包与发行 283
21.2.1 简介 283
21.2.2 打包的具体操作步骤 283
21.2.3 测试打包效果 285
21.3 用Ant来打包 285
21.4 本章小结 291

第22章 插件项目实战 292
22.1 前期准备工作 292
22.1.1 软件开发过程 292
22.1.2 本章项目开发环境的选择 293
22.1.3 安装MySQL 296
22.1.4 在Eclipse插件中连接MySQL数据库（版本V0001） 299
22.1.5 解决Java的中文问题 304
22.1.6 对字符集设置的测试结果 307
22.2 面向对象分析和数据表创建（版本V0010） 315
22.2.1 界面效果及实现功能 315
22.2.2 面向对象的分析与设计 316
22.2.3 创建数据表 325
22.2.4 给数据表插入数据 330
22.3 创建项目的主界面框架 332
22.3.1 前言 332
22.3.2 创建透视图及主功能视图（版本V0020） 332
22.3.3 创建“功能导航器视图”的树（版本V0020） 335
22.3.4 创建项目的图像注册表（版本V0030） 341
22.4 用户登录、退出功能的实现（版本V0040） 344
22.4.1 实现方案 344
22.4.2 界面部分的源代码 345
22.4.3 数据库部分的源代码 352
22.4.4 小结 357
22.5 “档案管理”编辑器的实现 358
22.5.1 前言 358
22.5.2 编辑器的创建与排序、翻页功能的实现（版本V0050） 358
22.5.3 实现删除用户功能（版本V0060） 372
22.5.4 实现新增用户功能（版本V0060） 374

22.5.5 实现修改用户的功能（版本V0070） 388
22.6 “成绩管理”编辑器的实现（版本V0080） 394
22.6.1 前言 394
22.6.2 单击结点打开视图 395
22.6.3 实现搜索视图SearchView 396
22.6.4 实现“成绩管理”编辑器 401
22.7 让软件适应多种数据库（版本V0090） 404
22.7.1 前言 404
22.7.2 解决方案 404
22.7.3 具体实现的源代码 405
22.8 首选项的实现（版本V0100） 407
22.8.1 前言 407
22.8.2 首选项的源代码 407
22.8.3 将程序中的设置值改成取之于首选项的设置 413
22.9 本章小结 414

第4篇　Web开发篇

第23章 Web环境的搭建（V0010） 416
23.1 概述 416
23.1.1 Eclipse的Web开发简介 416
23.1.2 本篇开发环境 416
23.1.3 CVS注意事项 416
23.2 Tomcat的下载与安装 417
23.2.1 下载Tomcat 417
23.2.2 安装Tomcat 418
23.2.3 启动Tomcat 419
23.3 Lomboz的下载与安装 420
23.3.1 下载Lomboz 420
23.3.2 安装Lomboz 421
23.4 Lomboz的环境设置 422
23.5 JSP的HelloWorld 423
23.5.1 设置Java的构建路径 423
23.5.2 创建一个J2EE项目 423
23.5.3 在Lomboz中启动Tomcat 424
23.5.4 发布hello模块 425
23.5.5 修改index.jsp 427
23.5.6 一些相关问题 427

23.6 如何不必发布就可以在IE上显示Web修改效果 427
23.7 配置Tomcat的数据库连接池 429
23.7.1 复制JDBC连接包 429
23.7.2 进入Tomcat的配置页面 429
23.7.3 修改Tomcat的server.xml文件 430
23.7.4 测试数据库连接池 431
23.8 本章小结 432

第24章 一个纯JSP+JavaBean实例（V0020） 433
24.1 JavaBean的环境配置 433
24.2 创建JavaBean及数据库层 434
24.2.1 AbstractDbOperate类 434
24.2.2 数据库连接管理器ConnectManager类 435
24.2.3 为SMS类增加一个字符串常量 436
24.2.4 修改工厂类SmsFactory 436
24.3 编写前台的JSP文件 436
24.3.1 示例的流程及效果 436
24.3.2 JSP文件说明 436
24.3.3 源代码 437
24.4 本章小结 441

第25章 在Eclipse中使用Struts 443
25.1 概述 443
25.2 Struts的下载及安装 443
25.3 Struts入门实例 446
25.3.1 Struts原理简介 446
25.3.2 用户登录实例 447
25.4 让Dreamweaver支持Struts标签 454
25.4.1 插件的下载 455
25.4.2 插件的安装 455
25.4.3 将struts标签添加到Dreamweaver中 455
25.4.4 插件的效果 456
25.5 深入struts-config.xml 456
25.5.1 页面转发 456
25.5.2 <form-beans>项之动态ActionForm 457
25.5.3 <action-mappings>项 458
25.5.4 使用DispatchAction类 459
25.5.5 使用多个struts-config.xml配置文件 460
25.5.6 解决命名冲突 460

25.6 验证的多种方法 461
25.6.1 方法一 461
25.6.2 方法二 462
25.7 使用更多的Struts标签 469
25.7.1 获知更多的标签 469
25.7.2 表单类标签 470
25.7.3 其他说明 472
25.8 本章小结 472
第26章 在Eclipse中使用Hibernate 473
26.1 概述 473
26.1.1 持久化框架产生的背景和现状 473
26.1.2 Hibernate简介 473
26.1.3 本章说明 474
26.2 Hibernate的下载和安装（V0050） 474
26.2.1 下载 474
26.2.2 安装 476
26.3 一个简单的Hibernate实例 478
26.3.1 创建XML映射文件：model.hbm.xml 478
26.3.2 创建管理session的类：HibernateUtil 479
26.3.3 创建一个用于测试的HibernateTest类 481
26.3.4 在hello目录下创建一个JSP文件：hibernateTest.jsp 482
26.3.5 总结 482
26.3.6 实践建议 483
26.4 继续深入使用Hibernate（V0060） 483
26.4.1 概述 483
26.4.2 编写XML映射文件 483
26.4.3 数据库操作类的实现 487
26.4.4 修改使用DbOperate类的程序 489
26.5 实现用户的修改、删除功能（V0070） 490
26.5.1 程序界面效果及功能说明 490
26.5.2 修改DbOperate类 491
26.5.3 修改用户列表userList.jsp文件 492
26.5.4 修改struts-config.xml 493
26.5.5 修改UserForm类 493
26.5.6 创建UserAction类 494
26.5.7 创建modifyUser.jsp 496
26.6 解决Tomcat的中文问题 497
26.7 Hibernate的自动生成工具 498

26.7.1　由XML映射文件生成数据库表 498
26.7.2　由实体类自动得到XML映射文件：class2hbm 499
26.7.3　由XML映射文件得到实体类：hbm2java 501
26.7.4　由数据库导出成XML映射文件：ddl2hbm.bat 502
26.8　本章小结 503

附录　光盘内容简介 504

第 1 篇

第 1 章　Eclipse 简介

第 2 章　安装 Eclipse 及多国语言包

第 3 章　安装 SWT Designer

第 4 章　Eclipse 的集成开发环境

第 5 章　Eclipse 中 CVS 的使用

第 1 章　Eclipse 简介

1.1　Eclipse 的历史

Eclipse 最初是 IBM 公司的一个软件产品，2001 年 11 月其 1.0 版正式发布。面世之初，Eclipse 还很不起眼，并饱受业界批评。所谓万事开头难，软件的初版一般都比较稚嫩。到 2003 年 3 月，Eclipse 发布了它的 2.1 版，立刻引起了轰动，下载的人蜂拥而至，导致它的下载服务器都因超载而拥塞了。

现在，IBM 已将投入巨资开发的 Eclipse 作为一个开源项目捐献给了开源组织 Eclipse.org，Eclipse 出色而有独创性的平台特性，吸引了众多大公司加入到 Eclipse 这个平台的发展上来，这样的公司有 HP、Oracle、Sybase、Borland、RedHat、Rational Software、SuSE、Together Soft 等。

最初 Eclipse 的开发人员，大部分都来自于当年与 JBuilder 齐名的 Visual Age 项目组，现在虽然 IBM 已将 Eclipse 捐献给了开源组织，但仍由 IBM 的子公司 OTI（主要从事 Eclipse 开发的人）继续领导着 Eclipse 的开发。

1.2　Eclipse 的体系结构

Eclipse 是一种通用工具平台——普遍适用的开放式扩展 IDE。它提供了功能丰富的开发环境，允许开发者高效地创建一些能够无缝集成到 Eclipse 平台中的工具。Eclipse 的体系结构图如图 1.1 所示。

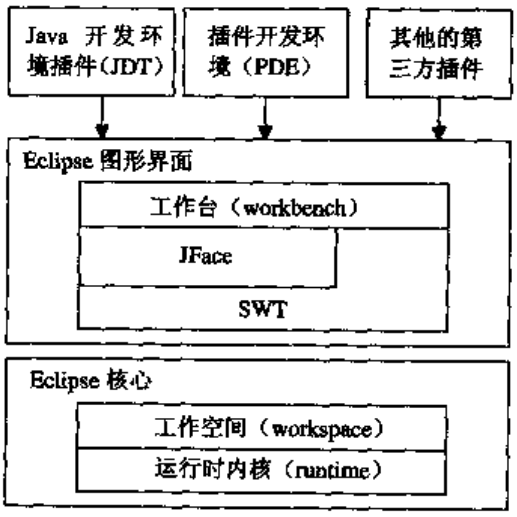


图 1.1　Eclipse 的体系结构图

Eclipse 的设计思想是：一切皆为插件。Eclipse 的核心是非常小的，其他所有的功能都以插件的形式附加到这个 Eclipse 核心之上。这样的功能插件包括它的图形 API（称为 SWT/JFace）、Java 开发环境插件（简称 JDT）、插件开发环境（简称 PDE）等。Eclipse 还对这些插件的协同工作提供了良好的支持，不仅安装简单，而且还可以无缝结合。

Eclipse 对这些插件是动态载入动态调用的。所谓动态就是指，Eclipse 启动后要真正用到这个插件时，它才会被调入内存，当插件不再被使用时，它就会被在适当的时候清除出内存。因此即使装了一大堆插件在 Eclipse 里，也不必担心某些不常用的插件白白占着内存。

1.3　优秀的图形 API：SWT/JFace

Eclipse 界面非常漂亮，如果不事先告之，仅从界面上是看不出任何 Java 程序的影子。图 1.2 是 Eclipse 在开发中的截图。

图 1.2　Eclipse 令人炫目的界面

Java 的标准图形 API：AWT/SWING 一直是被人诟病的，用它写成的界面不够美观，界面响应速度还慢。Eclipse 的 SWT/JFace 图形 API 彻底地改变了这一状况。主要原因是 SWT/JFace 与 AWT/SWING 在实现上有很大不同：AWT/SWING 是模拟本机窗口组件，而 SWT/JFace 则直接调用本机窗口组件，当本机没有所需组件时才进行模拟。

当然 SWT/JFace 这种实现方法也是有缺点的：它没有 Java 的跨操作系统的特性，每一个操作系统要用 Eclipse，都需要移植一次 SWT/JFace，这也是 SWT/JFace 最具争议的地方。不过因为 SWT/JFace 已经移植到了几个主流的操作系统平台，所以这已经不再是一个什么严重问题了。

这里需要强调两点：

- Eclipse 的界面虽然是基于 SWT/Jface 的，但除非是写 Eclipse 插件，否则仍然可以在 Eclipse 中使用 AWT/SWING 来开发标准的 Java 程序。
- 虽然 SWT/JFace 需要移植到每一个操作系统平台，但基于 SWT/JFace 开发的 Java 程序并不需要移植，因为 SWT/JFace 已经为各操作系统编程提供了相同的接口。

1.4　开放式、可扩展的 IDE

Eclipse 有极为强大的集成开发环境，它集成了 CVS、JUnit 和 Ant，并且强大的代码重构功能独步江湖无人能及。因为 Eclipse 的安装包集成了 Java 开发环境的插件 JDT，所以 Eclipse 默认是一个和 JBuilder 类似的 Java 开发工具。但 Eclipse 不仅仅只是 Java 的开发工具，我们只需要装上 C/C++的插件（简称 CDT），就可以把它当作一个 C/C++开发工具来使用。只要有相应的插件，Eclipse 也可以用做其他语言的开发工具。

Eclipse 的插件扩展机制是其最突出的特点和优势，它使 Eclipse 提升到了一个平台的高度。我们可以利用 Eclipse 的插件开发环境（简称 PDE）来开发自己的 Eclipse 插件，随己所需地扩展 Eclipse 的功能。这样的插件形式是多种多样的，它可以是一种编程工具，如 C/C++、JSP、PHP，也可以是一个应用系统，如信息管理系统。以下就是一个用 Eclipse 插件方式开发的一个信息管理系统，如图 1.3 所示。

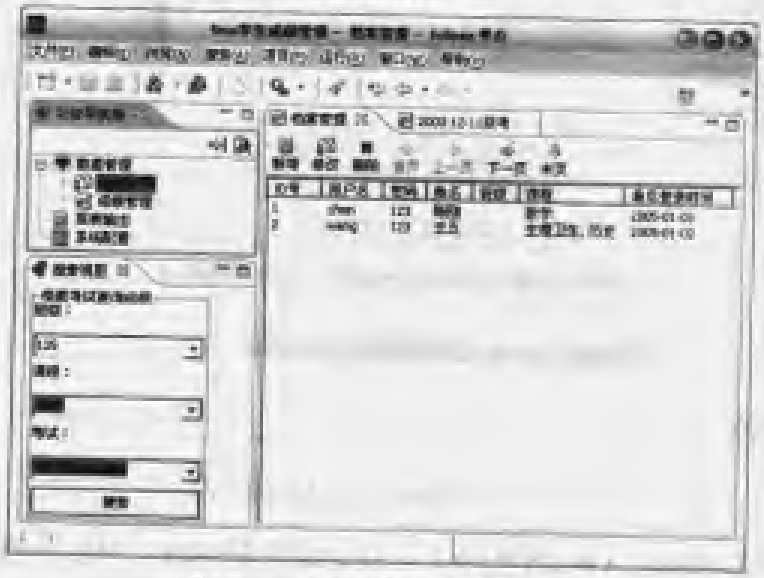


图 1.3　一个信息管理系统的截图

Eclipse 的插件机制使得 Eclipse 具有无限扩展的可能，Eclipse 就像一个精巧设计好的插花基座，我们可以在上面插上玫瑰、菊花、满天星，将 Eclipse 打扮得分外妖娆。

1.5　全中文化的帮助文件

Eclipse 的帮助非常全面，而且给 Eclipse 装上多国语言包后，其帮助也全部中文化，如

图 1.4 所示。从其帮助中读者可以很快地熟悉 Eclipse 的使用，这也是本书不把 Eclipse 集成环境的使用作为重点内容讲解的主要原因，因为 Eclipse 的帮助文档讲得已经很好很全了。

极少有国外的 Java 开发工具能把帮助文件也汉化成中文的，就是将软件本身汉化成中文的也很少。从这一点可以看到 Eclipse 对于中国用户的友好性。

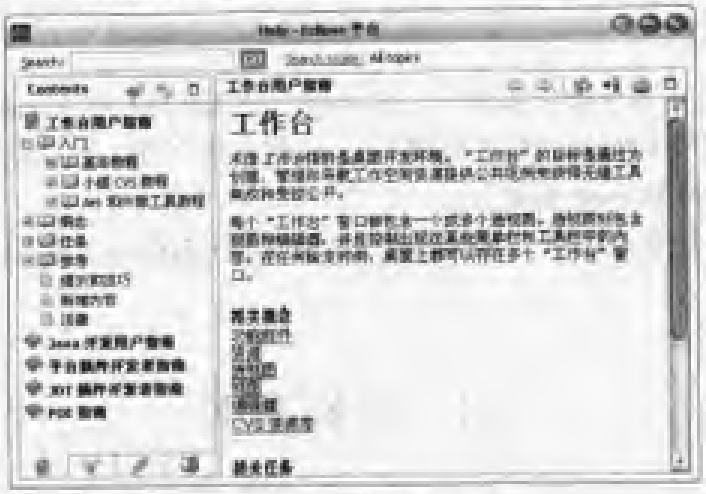


图 1.4　全中文化的帮助

1.6　Eclipse 是开放源代码的

现在很多大型公司的开发部门都由 JBuilder 转向了 Eclipse，主要原因就是 Eclipse 是免费的。而对于程序员来说，Eclipse 的开放源代码让他们可以阅读最顶级的 Java 源码，学习最优秀的 Java 技术。在 Eclipse 源码中，有着诸多设计模式的使用，其编码的优雅和高效真是令人叹服不已，而且在 Eclipse 中阅读这些源代码是非常方便的。

就国内现状来说，Eclipse 主要还是作为编程工具来使用。将 Eclipse 作为一个平台并在此平台上开发插件和系统的个人或公司还是非常少。其实，Eclipse 的插件开发并不复杂，主要原因还是太缺乏 Eclipse 插件开发的资料。

由于是基于 Eclipse 平台开发的软件，其界面美观度和响应速度是普通 AWT/SWING 程序所无法比拟的，并且以 Eclipse 插件方式来开发系统，可以基于原有 Eclipse 菜单、编辑器来扩展，节省了很多开发精力，所以 Eclipse 的插件开发在中国有着很广的应用前景以及极大的潜在市场需求。

1.7　本 章 小 结

本章介绍了 Eclipse 的历史和一些技术特征及优点，通过学习本章的内容，希望能使读者对 Eclipse 有一个概括性的了解，在第 2 章开始学习如何安装 Eclipse。

第 2 章　安装 Eclipse 及多国语言包

Eclipse 可以安装在各种操作系统上，本书仅介绍 Eclipse 在 Windows 操作系统下的安装。在 Windows 下安装 Eclipse，除了需要 Eclipse 软件包之外，还需要 Java 的 JDK（或 Java 运行环境 JRE）来支持 Eclipse 的运行。

本书所涉及的软件安装环境：Windows XP 中文版+JDK1.4.2_06+Eclipse 3.0.1（加多国语言包）。

2.1　安装 JDK

去 SUN 公司的网站下载 JDK。具体下载地址：http://Java.sun.com/j2se/，选择 1.4 版中的最新版来下载（本书使用 JDK1.4.2_06）。具体的 JDK 下载方法参阅配书光盘的视频演示。

（1）运行下载的 JDK 的 exe 安装文件。

（2）在弹出的对话框中选择 I accept the terms in the license agreement 选项，同意协议内容，然后单击 Next 按钮继续下一步。

（3）默认的就是安装所有的 JDK 组件，不必改变它。但将默认的安装路径改为“C:\jdk”，然后单击 Next 按钮继续下一步。

（4）默认已经选中“Microsoft Internet Explorer”选项，不必改变它，直接单击 Next 按钮，开始安装 JDK。

在安装 JDK 的时候需要注意：

- 在 Windows XP 下安装 Eclipse 不需要设置 JDK 的环境变量。实际上本书从始至终都没有设置它。请注意本书的开发环境，有可能在 Windows 2000、Linux 下或安装其他版本的 JDK，需要设置 JDK 的 path、classpath 等环境变量。
- 如果 Eclipse 提示找不到 JDK 等错误，这时可以选择“控制面板→系统→高级→环境变量”选项，然后在 path 变量后加上 c:\jdk\bin，再创建一个 classpath 变量值为 c:\jdk\lib\tools.jar;c:\jdk\lib\dt.jar;。
- 最小只需要 Java 的运行环境 JRE 即可支持运行 Eclipse，本书之所以安装 JDK，是因为 JDK 已经包含了 JRE，同时又包含了 Java 的源代码，在 Eclipse 中可以很方便地查看这些源代码，在调试程序的时候尤其有用。
- JDK 或 JRE 需要选用 1.4.1 以上的版本，而且 Eclipse 3.0 还不支持 JDK 1.5。
- Eclipse 会尝试依次按如下 3 个地方查找 JRE：首先去启动参数-vm 设定的路径查找；如果没有-vm 参数，则查找安装目录下的 jre 子目录（本例就是 C:\eclipse3.0.1\eclipse）；如果还没有，Eclipse 将查找当前 Windows 系统默认的 JRE。因此如果

读者的系统 JDK 的版本太过于混乱则导致 Eclipse 无法启动，可以尝试以下方法解决：在命令行下执行 eclipse -vm c:\jdk\jre\bin\javaw；或者将 java 的 jre 目录复制到 C:\eclipse3.0.1\eclipse 目录下。

2.2　安装 Eclipse

去 Eclipse 的官方网站（www.eclipse.org）下载 Eclipse 软件包。本书选择安装 3.0.1 版本的 Eclipse。具体下载地址：http://download.eclipse.org/downloads/。下载方法参阅配书光盘的视频演示。

Eclipse 的安装是非常简单的，它属于绿色软件的安装方法：不需要运行安装程序，不需要往 Windows 的注册表写出信息，只需要将下载的 Eclipse 压缩包解压就可以运行 Eclipse 了。解压后复制到 C 盘，目录结构如图 2.1 所示。

图 2.1　Eclipse 安装的目录结构图

注意：Eclipse 安装包解压就可以运行 Eclipse，并且不一定要安装到 C 盘，但为了与本书同步，强烈建议保持 Eclipse 的安装目录结构和本书相同，以避免以后不必要的麻烦。

现在试运行一下 Eclipse，检查它是否安装成功了。双击 c:\eclipse\eclipse.exe 文件来启动 eclipse。

第一次启动时，会弹出一个对话框，如图 2.2 所示，在该对话框设定工作空间的路径，接受其默认路径就行了，并且将下面选项选上，这样以后启动 Eclipse 就不会再弹出该对话框。单击 OK 按钮，开始运行 Eclipse 程序。运行完毕后，出现一个 Welcome to Eclipse 3.0 的欢迎界面，表示 Eclipse 已经安装成功了。

注意：工作空间 workspace 目录中包含我们所编写项目的源代码，以及一些 Eclipse 个性

化的配置文件。如果要备份自己的软件项目，只要复制该目录即可。

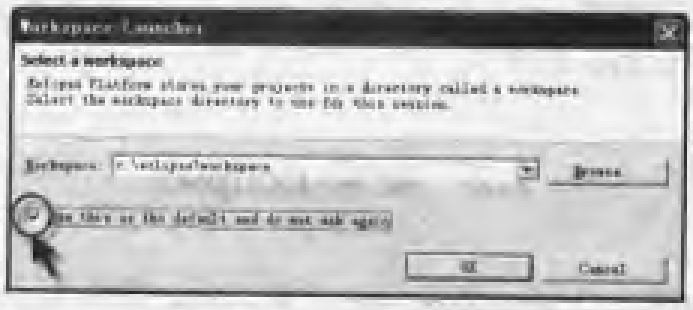


图 2.2　Workspace Launcher 对话框

如果出现如图 2.3 所示的对话框，表示安装失败，这时请检查安装的 JDK 版本是否正确。如果你的电脑里安装了太多的 JDK 版，建议只保留一个。

图 2.3　Eclipse 运行错误画面

当启动 Eclipse 之后，Eclipse 会在 c:\eclipse 目录下创建两个目录：workspace 和 configuration。workspace 在上面已经介绍过；configuration 则包括一些 Eclipse 自己的配置信息，它是可以任意删除的，不会影响到项目开发，并且 Eclipse 再启动时会再次创建该目录。

2.3　Eclipse 多国语言包的安装

Eclipse 多国语言包中不仅有 Eclipse 的中文翻译，同时也包含了其他几种主要语言的翻译，Eclipse 能够自动根据 Windows 操作系统的语言环境来选择使用语言包中的那一种语言，很智能化。

先去 Eclipse 的官方网站（www.eclipse.org）下载 3.0.1 版 Eclipse 的相应的多国语言包（网页上的英文名称为 Language Pack）。具体下载地址：http://download.eclipse.org/downloads/。下载方法可参阅配书光盘的视频演示。安装方法如下：

（1）先关掉 Eclipse。

（2）将下载的语言包（ZIP 文件）解压缩，并给这个目录取个名，本书取名为 language。

（3）将此目录复制到 C:\eclipse 下，复制完后的整个 Eclipse 目录结构如图 2.4 所示。注意，language 目录下有一个 eclipse 子目录，然后才是 features、plugins 目录。

（4）建立语言包的 link 文件。在 C:\eclipse 下新建一个目录并取名为 links，在该目录中新建一个文本文件并取名为 language.link（可任意取名），然后在 language.link 文件中加一行文字：

```
path=C:\eclipse\language　　（注意：路径分隔符由两个反斜杠组成）
```

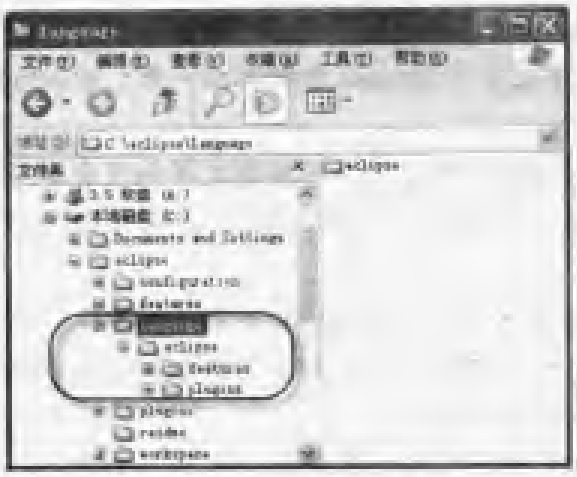


图 2.4　Eclipse 的目录结构图

注：关于路径的写法还有另外两种：

```
（1）path=C:/eclipse/language　　（用斜杠）
（2）path=language　（用相对路径）
```

经过以上四步之后，现在可以重新启动 Eclipse 了。Eclipse 启动的时候会自动查找 links 目录中的所有文本文件，并将其中 path 指向的插件加载到 Eclipse。启动后的界面如图 2.5 所示，打开 Eclipse 的帮助，可以发现帮助也已经是中文的了。

图 2.5　Eclipse 的中文界面

注意：

（1）所有的 link 文件必须在名为 links 的目录中才会有效。

（2）link 文件的文件名及扩展名可以任取，例如 myplugin.txt，language.ini 都可以。

（3）如果要安装多个插件，既可以将所有插件的 path 设定放在一个 link 文件中（一个 path 一行），也可以分成一个 path 一个 link 文件。

（4）如果安装了多国语言包后 Eclipse 还是没有中文化，应作如下检查：link 文件中 path 设定的路径是否正确；link 文件是否放在名为 links 的目录下；建立 links 目录的位置是否正确；多国语言包的版本是否与 Eclipse 的版本相一致；Windows 操作系统是不是中文版的。

（5）安装了多国语言包后，发现还有一些英文单词没有翻译成中文，这是因为之前启动了英文版的 Eclipse，Eclipse 在启动后，记录了一些英文版的信息，形成了干扰，解决办法有以下两种。

第一种：将 C:\eclipse 路径下的 configuration 目录删除掉，再重新启动 Eclipse。

第二种：右击 eclipse 快捷方式，在目标栏的最后加入 eclipse 的启动参数-clean。或者在 DOS 下用“eclipse-clean”来启动。

（6）怎么样将 Eclipse 从中文版还原回英文版呢？ Eclipse 是根据 link 文件里的路径设定来加载多国语言包插件的。因此删除、改名 links 目录，或者是删除语言包对应的 link 文件，都可以使 Eclipse 在启动时找不到多国语言包插件。但这时会出一个错误对话框，解决办法是将 C:\eclipse3.0.1\eclipse 路径下的 configuration 目录删除掉，再重启 Eclipse，就能返回为英文版的 Eclipse 了。

Eclipse 的多国语言包是 Eclipse 的一个插件，所以它的安装方法也就是插件在 Eclipse 中的安装方法。插件的安装方法有两种：

- 建立 link 文件的方式，多国语言包的安装属于这种方式。
- 将插件文件复制到 Eclipse 相应的 plugins、features 目录中，接下来的 SWT Designer 插件的安装使用这种方式。

这两种安装方式，第一种结构清楚、容易卸载，第二种只需要复制文件，最简单。但无论是哪一种方式，都可看到在 Eclipse 中安装插件是很方便的，而且当要卸载某个插件时，也不用担心它还会留着什么垃圾文件在系统里“阴魂不散”。

2.4 本章小结

本章主要介绍了 JDK 的安装、Eclipse 及多国语言包的安装，这三者的安装是步步深入互为基础的，而且 Eclipse 安装是否成功直接关系到后面的学习能否顺利进行。

第 3 章　安装 SWT Designer

SWT Designer 是一个很好的 Eclipse 的 SWT 界面开发插件包，SWT Designer 支持界面组件拖拉操作，有了它做起界面设计来就轻松多了。但千万别指望它能完成所有的界面设计工作，现在它存在的问题还很多。例如，当界面组件多起来的时候，它会引起计算机运行缓慢并将内存消耗殆尽而导致 Eclipse 报错退出。

虽然 SWT Designer 还有很多不足，但在 Eclipse 的 SWT 界面开发中，SWT Designer 已经为我们简化了很多界面设计的初期工作。在 Eclipse 中，手工写界面代码的能力是非常重要而且非常有用的，现在我们还无法奢望 Eclipse 中有一种界面设计工具能像 VB、Delphi 的那样好用。

3.1　下　　载

SWT Designer 的下载地址是：http://www.swt-designer.com/，不同的版本 Eclipse 需要使用相应版本的 SWT Designer，因为本书使用 Eclipse3.0.1 版，所以下载 SWT Designer 的版本为 Edition for Eclipse 3.0 GA & 3.0.1，如图 3.1 所示。下载方法参阅配书光盘的视频演示。

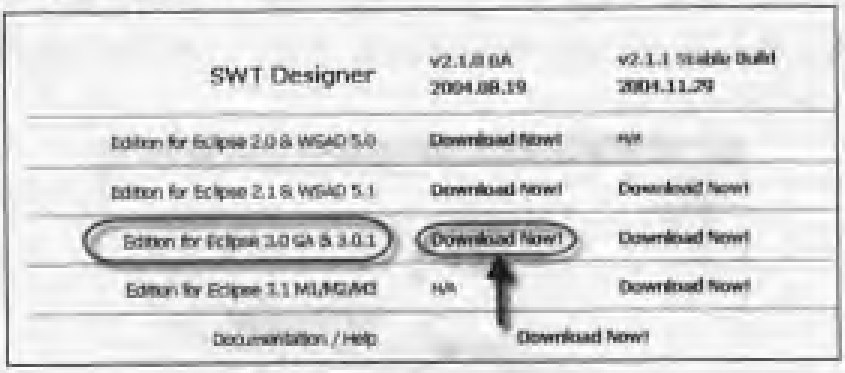


图 3.1　SWT Designer 下载列表

3.2　安　　装

SWT Designer 具体的安装步骤如下：

（1）将下载的安装文件解压缩。

（2）将解压缩后的目录 plugins 下的两个子目录复制到 Eclipse 的 plugins 目录中。

只需要用以上两步就完成了安装，Eclipse 插件的安装就是这么简单。然后检验一下是否安装成功了。打开 Eclipse 主菜单，选择“文件→新建→其他”选项，如果在弹出的“新

建项目”对话框中多了一个“Designer 项”，则表示 SWT Designer 安装成功，如图 3.2 所示。

图 3.2　“新建项目”对话框

3.3　注 册 激 活

真正使用 SWT Designer 前还需要像 Windows XP 一样注册激活。其步骤如下：

（1）单击主菜单，选择“窗口→首选项”选项，弹出一个对话框。

（2）单击对话框左边列表中的 Designer 选项，然后单击对话框右边的 Registration and Activation 按钮，如图 3.3 所示。

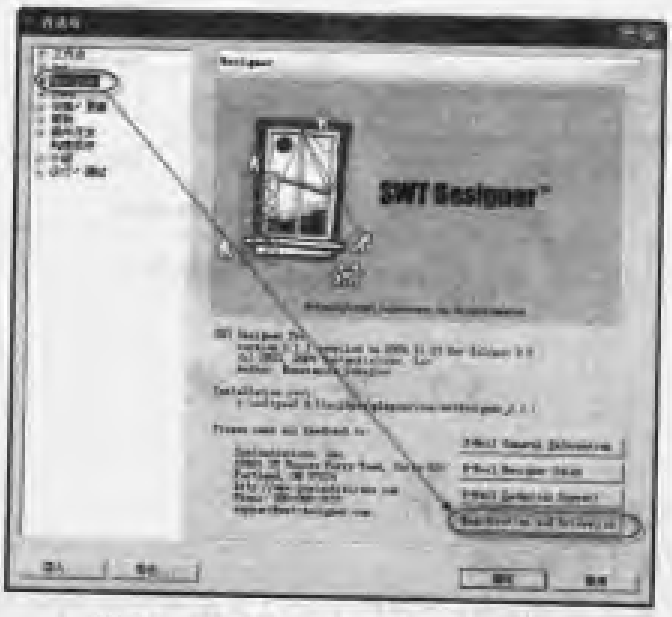


图 3.3　SWT Designer 的注册激活窗口

（3）在弹出的 SWT Designer 注册激活窗口中，选择 SWT Designer 和 Free（limited functionality）选项，然后单击“下一步”按钮。（注意，选择 SWT Designer 选项是因为本书内容只涉及 SWT/JFace 的使用；选择 Free （limited functionality）选项，是因为 Free 版免费且功能已经足够开发使用了。）

（4）如图 3.4 所示，在新弹出的窗口中填写相关注册信息，然后单击“下一步”按钮。

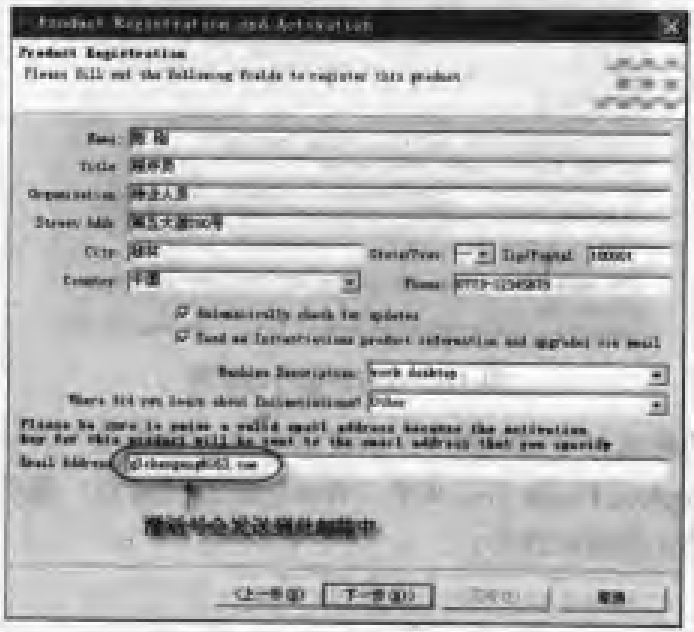


图 3.4　“注册信息填写”窗口

（5）在弹出的窗口中接受默认的选择（Directly over web 项），直接单击“下一步”按钮，这时出现一个错误提示框，上面的文字说明是：激活号（activation key）已经发到你的邮箱，如图 3.5 所示。

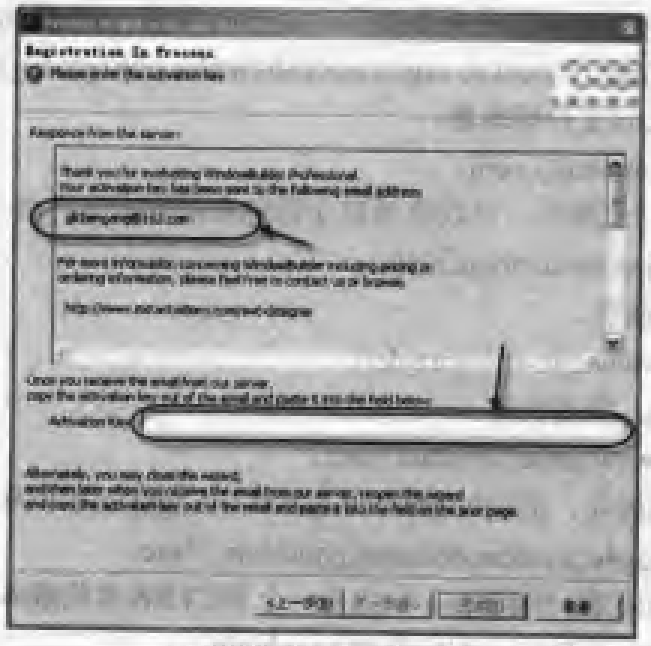


图 3.5　填写激活号的窗口

（6）去邮箱打开这封主题为 Online Activation: SWTDesignerEval 的邮件（如果网络不堵塞，邮件应该马上就能收到），激活号已经包含其中，如图 3.6 所示。

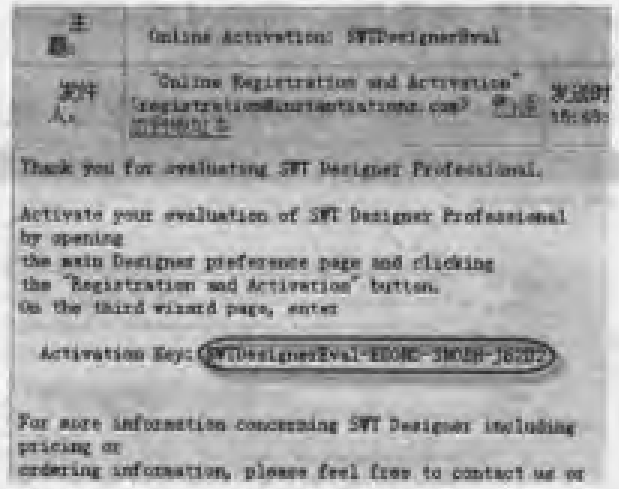


图 3.6　含有激活号的邮件

（7）将邮件中的激活号复制到图 3.5 的文本框中，并单击“下一步”按钮。如果界面出现 Activation is complete.Thank you.文字，表示已经激活成功。最后单击“完成”按钮，结束 SWT Designer 的注册激活操作。

3.4　其他常用插件

- MyEclipse：J2EE开发插件，支持JSP、EJB、数据库等操作。
  下载网址：www.myeclipseide.com。
- Lomboz：和MyEclipse的同类型插件。Lomboz免费，MyEclipse收费。
  下载网址：http://forge.objectweb.org/project/showfiles.php?group_id=97。
- XMLBuddy：xml文件编辑器。
  下载网址：www.xmlbuddy.com。
- Fat Jar：项目打包插件，可以将项目支持包和项目本身打成一个包。
  下载网址：http://fjep.sourceforge.net/。
- Jinto：国际化插件。
  下载网址：http://www.guh-software.de/。
- JasperAssistant：报表插件。
  下载网址：http://www.jasperassistant.com/。
- Log4E：Log4j插件，提供Log4j的快速操作。Log4j专用于为程序输入调试信息。
  下载网址：http://log4e.jayefem.de/index.php/Main_Page。
- VSSPlugin：VSS客户端插件，VSS是一个和CVS齐名的版本管理系统。
  下载网址：http://sourceforge.net/projects/vssplugin。
- Implementors：当追踪方法代码时，Eclipse默认是转到方法的接口类，而接口中是

只有方法名没有代码的。此插件提供了追踪到方法的实现代码的功能，比较实用。

下载网址：http://eclipse-tools.sourceforge.net/implementors/。

- Call Hierarchy：显示一个方法的调用层次，可以从中看到它被哪些方法调用了，以及它调用了哪些方法，是代码追踪比较实用的工具。

下载网址：http://eclipse-tools.sourceforge.net/call-hierarchy/。

- Hibernate Synchronizer：Hibernate插件，提供Hibernate的自动映射等操作。

下载网址：http://www.binamics.com/hibernatesync/。

- Profiler：性能跟踪、测量工具，能跟踪、测量B/S模式开发的程序。

下载网址：http://sourceforge.net/projects/eclipsecolorer/。

3.5 本章小结

本章主要介绍了 SWT Designer 插件的安装，其安装很简单，麻烦的是激活。但 SWT Designer 对学习后面章节的 SWT、JFace 内容帮助很大，应该把它安装成功。在最后还给出了一些常用插件的下载地址，这些插件对开发都是很有帮助的。

第 4 章　Eclipse 的集成开发环境

Eclipse 开发环境的易用性很好，而且又有中文化的界面，以及详细的中文帮助。全面掌握 Eclipse 开发环境的使用并不困难。在此，本章先简单介绍了 Eclipse 界面中的各个部分，然后精选了一些最常用的使用技巧来讲解，以期让读者能够快速上手。

4.1　Eclipse 界面简介

本节以 Eclipse 启动时默认的“资源透视图”（如图 4.1 所示）来讲解 Eclipse 的集成开发环境。

图 4.1　资源透视图

图中的整个窗口称为 Eclipse 的工作台，其主要有以下几个组成部分：菜单栏、工具栏（tool bar）、透视图（perspective），而透视图又分为视图（view）、编辑器（editor）。图中已经标出了各个部分的位置。其中，透视图和视图是 Eclipse 界面中比较重要的概念。

视图是 Eclipse 中一个个功能窗口，例如在如图 4.1 所示的界面中：

- 导航器视图：显示项目中的文件列表。
- 大纲视图：显示当前编辑器所打开文件的纲要。单击“大纲视图”中的各项可以在文件中快速定位。
- 任务视图：显示代码中未完成的任务。在代码中加一条带 TODO 的注释，就可以在任务视图中添加一项任务。

透视图则是由一些视图、编辑器组成的集合。打个比喻：把鹿的角、蛇的身、鱼的鳞、鹰的爪等组合在一起就成了“龙”，鹿角、蛇身等就相当于一个个视图或编辑器，而龙就相当于一个透视图。而且透视图中的视图并非是一成不变的，可以使用鼠标拖动操作更改各视图的位置和大小，也可以关闭一些视图或加入一些其他的视图，Eclipse 能自动记忆下这些改变。

Eclipse 绝大部分功能都体现在视图中。除了上面介绍的几个视图外，Eclipse 还有很多视图。选择主菜单“窗口→显示视图→其他”选项，可弹出如图 4.2 所示的包含所有视图的窗口。选择主菜单“窗口→打开透视图→其他”选项，可弹出如图 4.3 所示的包含所有透视图的窗口。

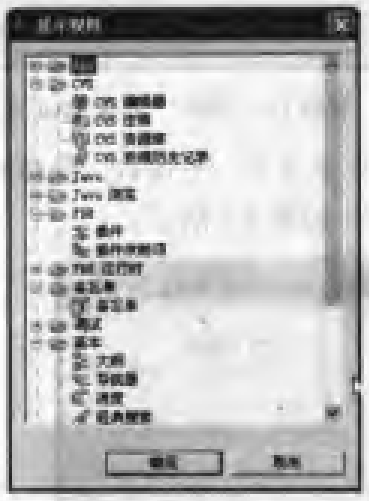


图 4.2　“显示视图”窗口

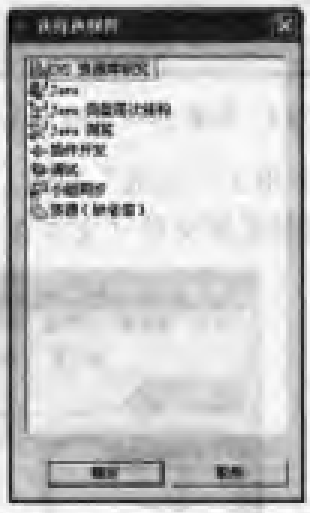


图 4.3　“选择透视图”窗口

注意：

（1）通过选择主菜单“窗口→复位透视图”选项可以将当前透视图还原为其初始界面。

（2）可以定制自己喜欢的视图布局组合，并能将布局组合另存为一个新的透视图。方法是：单击主菜单“窗口→将透视图另存为...”选项。

4.2　创建 Java 项目并运行

Java 项目包含用于构建 Java 程序的源代码和相关文件。创建一个 Java 项目主要分为以下几步。

第一步：新建一个空 Java 项目。

（1）选择主菜单“文件→新建→项目”选项，在弹出的窗口中选择“Java→Java 项目”选项，如图 4.4 所示。

（2）单击“下一步”按钮，出现如图 4.5 所示的窗口。在“项目名称”文本框中输入 myproject，工作空间 workspace 中将会建立一个同名目录。项目布局选择“创建单独的源文件夹和输出文件夹”选项，这样项目中会生成两个目录：src 和 bin，前者用来存放源代码，后者用来保存编译后的 class 文件。

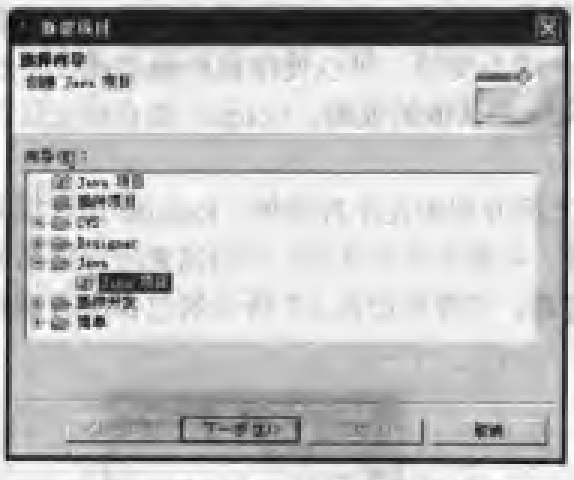


图 4.4　新建 Java 项目—选择向导

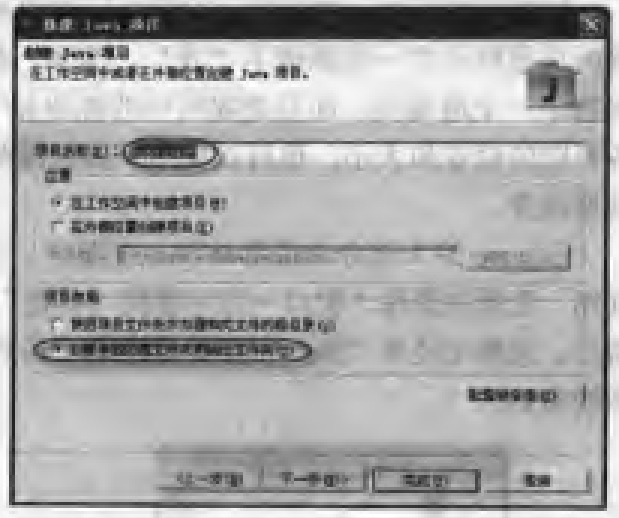


图 4.5　新建 Java 项目—创建 Java 项目

（3）单击“完成”按钮，就会看到在“导航器视图”中多了一个项目，此项目最初只有两个项目配置文件（文本文件）和两个空目录 src、bin，如图 4.6 所示。

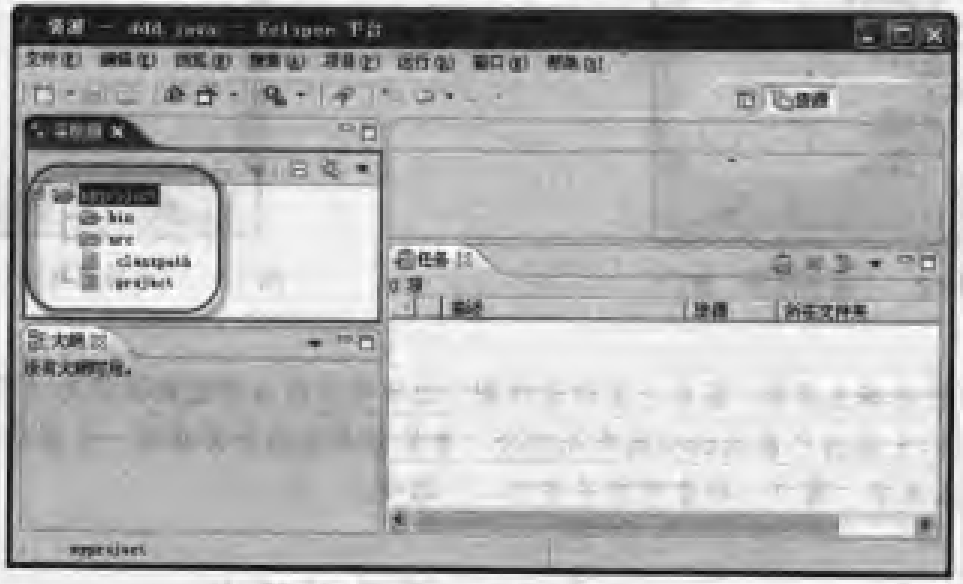


图 4.6　导航器视图

第二步：新建一个 Java 类。

（1）单击如图 4.7 所示小圈中的工具栏按钮，并选择“其他”选项。在弹出的“新建”窗口的左部列表中选择“类”这一项。（也可以按 Ctrl+N 键打开“新建”窗口）。

图 4.7　“新建”工具栏按钮

（2）在弹出的"新建 Java 类"窗口中输入类名 HelloWorld，输入包名 net.yeah.glchengang（包名通常用倒写的网址来起名，这样可以保证此包名是世界惟一的），并选中 public static void main(String[] args)选项来自动创建一个 main 方法，如图 4.8 所示。

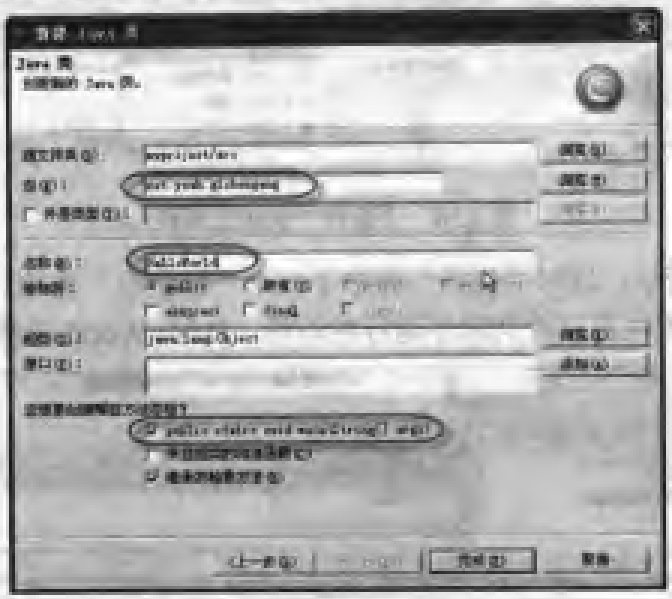


图 4.8　"新建 Java 类"窗口

（3）单击"完成"按钮后，一个代码编辑器被打开，如图 4.9 所示。可以看到左边 src 目录多了几个和包名相对应的子目录，而 bin 目录结构则与 src 自动保持同步。编辑器窗口的代码多了很多注释，这些注释都是可以自定义的（详见 4.3 节）。

（4）如图 4.9 所示，在编辑器中加入一行代码 System.out.println("Hello World");。

图 4.9　建立项目及类之后的界面

第三步：运行项目。

（1）如图 4.10 中的圈所示，单击按钮旁的小黑三角形，选择"运行（R）为→Java 应用程序"选项来运行 HelloWorld 程序。以后再运行 HelloWorld 程序就不用这么麻烦了。

直接单击按钮即可。

图 4.10　“运行”快捷工具栏

（2）如图 4.11 所示，这时“任务”旁多了一个“控制台”选项，单击“控制台”转到控制台视图，在该视图中可以看到程序运行的结果，Hello World 已经显示其上。

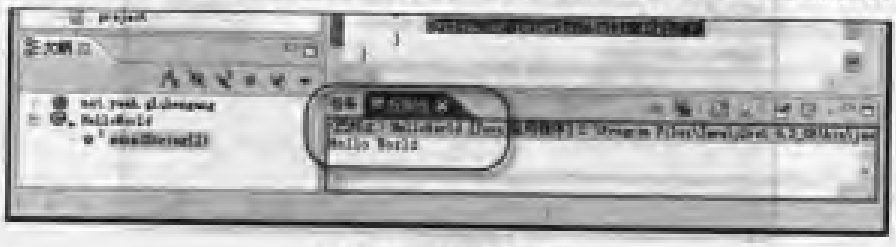


图 4.11　控制台视图

4.3　自定义注释及代码格式化

4.3.1　自定义注释

如果自动生成的代码注释，并非我们想要的，那么可以重新设定注释。具体步骤如下：

（1）单击主菜单“窗口→首选项”选项，弹出如图 4.12 所示的窗口。

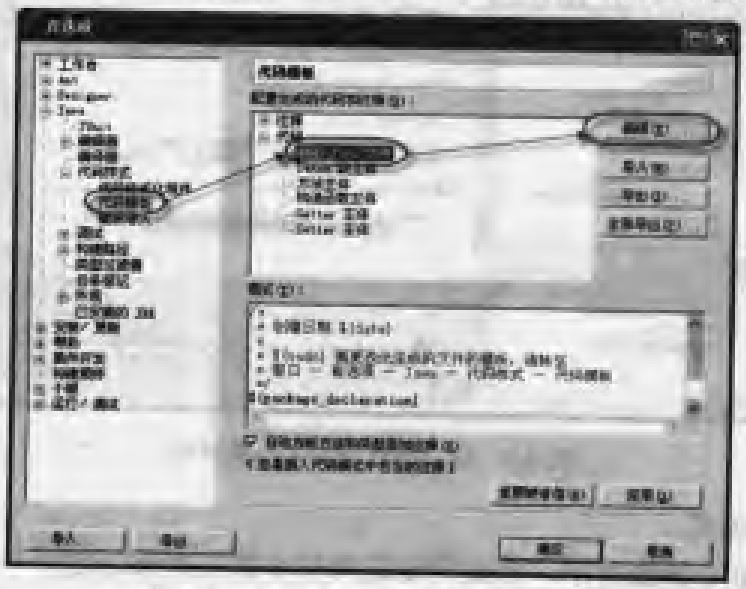


图 4.12　首选项的“代码模板”窗口

（2）选择“Java→代码样式→代码模板”选项，再选择“代码→新的 Java 文件”选项，最后单击“编辑”按钮，弹出如图 4.13 所示的窗口。

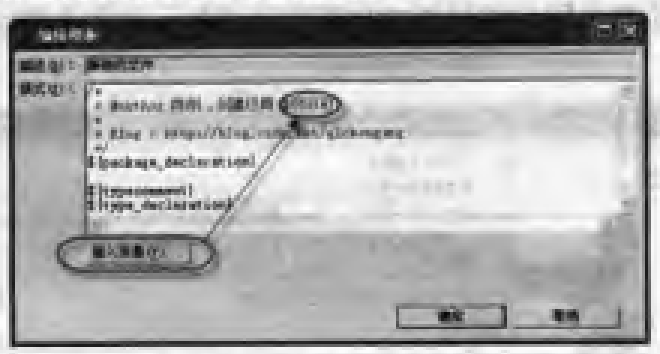


图 4.13　“编辑模板”窗口

（3）将原来的注释改成图 4.13 中的文字。单击“插入变量”按钮便可以在窗口中插入一些变量（如${date}），在代码中生成注释时，这些变量会被相应的值替换。

（4）按照 4.2 节所讲的方法再建一个新的类，如图 4.14 所示，可以看到代码开头的注释已经变为我们自定义的注释。

图 4.14　自定义的注释

自定义注释的功能样式很多，这里仅介绍最常用的一种，其他的操作方法与此类似。

4.3.2　代码格式化

每个人写代码的风格和习惯都不一样，例如，有人喜欢把首括号写在第一行的后面，有人则喜欢将首括号另起一行，还有些人则天马行空，代码毫无章法可言。不过没关系，Eclipse 为代码的格式化提供了强大的支持，在代码编辑器中按键 Shift+Ctrl+F，代码就会变得很整齐规范了。另外，还可以改变代码格式化的设置。具体操作步骤如下：

（1）单击主菜单“窗口→首选项”选项，弹出如图 4.15 所示的窗口。

（2）选择“Java→代码样式→代码格式化程序”选项，再单击“显示”按钮。弹出如图 4.16 所示的代码格式化设置窗口，窗口右边部分的示例代码可以实时反映当前设置的效果。

（3）设置完成后，单击“确定”按钮，在弹出的窗口中为当前的设置输入一个新的名称。

4.3.3　实践建议

项目开发团队的代码格式化及注释都应该保持高度一致。因为一般项目开发都会使用

CVS 或 VSS 来管理源代码。如果程序员从 CVS 上更新他人代码，并且再次格式化代码后，如果两人格式化设置不同，则 CVS 会显示该程序有很多地方都被修改了，但也许真正修改的地方只有一行，其他都是因为代码格式化不同造成的。这样会极大影响对前后版本的修改变化的查阅。

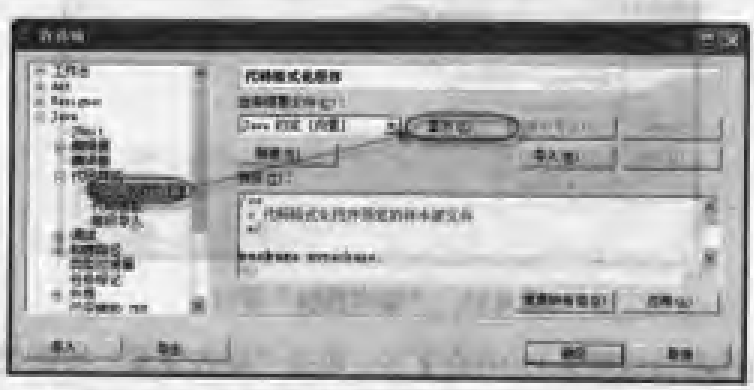


图 4.15　首选项的“代码格式化程序”窗口

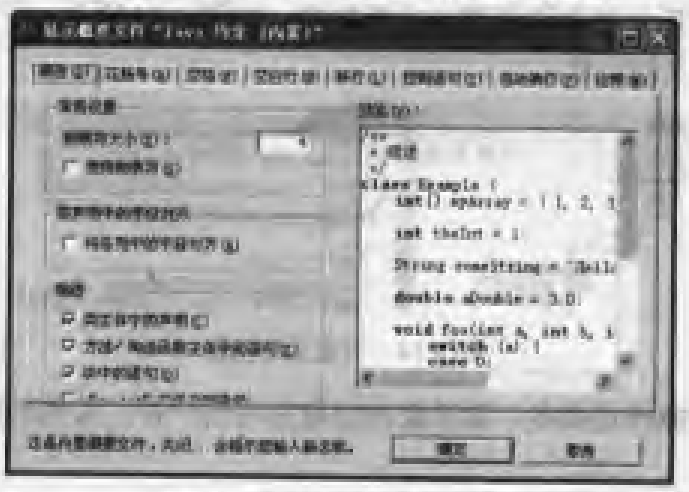


图 4.16　代码格式化的设置窗口

4.4　使用 Eclipse 的重构功能

Eclipse 的重构功能是笔者最喜欢、也是最常用的一个功能。在一个复杂的项目系统中，存在继承、联合等繁杂的应用和交错使用的代码，而编程中代码调整也是很频繁的工作。例如，要把某变量改成一个更合适的名称，也许就要涉及改动几十个地方；如果要改类名就更痛苦了，往往要慢慢回忆在哪些 Java 文件中引用了这个类，然后一个个去修改。

Eclipse 重构中的“重命名”功能为我们解决了变量、类、方法、包等重命名的所有问题。如图 4.17 所示，要将变量 a 改为 i，手工要改七处，但用 Eclipse 重构的“重命名”功能，只需要改动一处即可。具体操作步骤如下：

（1）将 Abc.java 的编辑框最大化，以方便操作，将变量 a 选择上。

（2）在编辑器中单击右键，选择“重构→重命名”选项。

（3）在弹出的窗口中，填写新变量名称 i，单击“确定”按钮完成重命名操作。

注意：如果单击“预览”按钮，可以在更改之前预览修改的效果。另外，如果修改引起代码冲突，Eclipse 会出现一个提示冲突情况的窗口，我们可以选择继续应用更改，或者取消更改操作。

图 4.17　重构

本节仅演示了变量的重命名。类、包、方法的重命名也与此类似。除此之外，Eclipse 还能自动侦测到类、包的剪切、移动，并能自动完成代码改动。

4.5　任 务 标 记

4.5.1　任务标记的设定

在编写程序时有一些工作要等到以后才做，这时可以在代码加上一行以//TODO 为开头的注释，来创建一条任务标记，以后双击此任务标记就可以快速定位到代码注释处。

Eclipse 3.0 版默认的任务标记有三个：//FIXME、//TODO、//XXX，它们有着不同的优先级。在多人项目中，最好每人定义一个属于自己的任务标记，以方便和其他人的区别开来。具体操作步骤如下：

（1）单击主菜单“窗口→首选项”选项，再选择“Java→任务标记”选项，然后单击“新建”按钮，操作如图 4.18 所示。

（2）如图 4.19 所示，在弹出的窗口中输入自己的标记，建议用大写字母取名（本书为 CHEN_TODO）。再选择优先级，建议选择“高”选项，然后单击“确定”按钮。

（3）返回如图 4.18 所示的窗口后，单击的窗口的“确定”按钮后又会弹出一个对话框，询问是否为了应用新标记而重编译所有程序，单击“是”按钮，Eclipse 开始编译项目。

自定义的效果如图 4.20 所示，优先级高的任务旁会有一个感叹号。

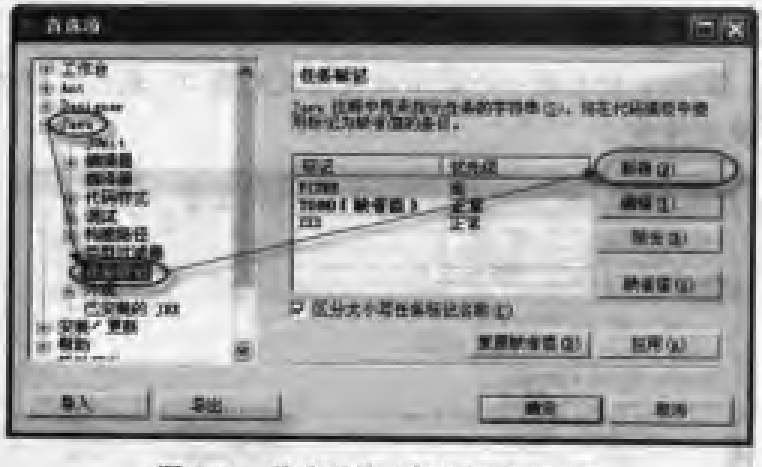


图 4.18　首选项的“任务标记”窗口

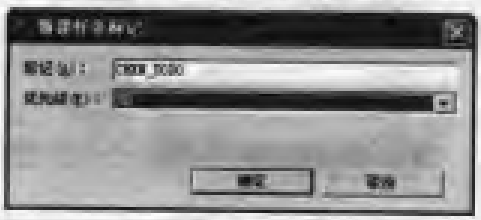


图 4.19　设定任务标记名称

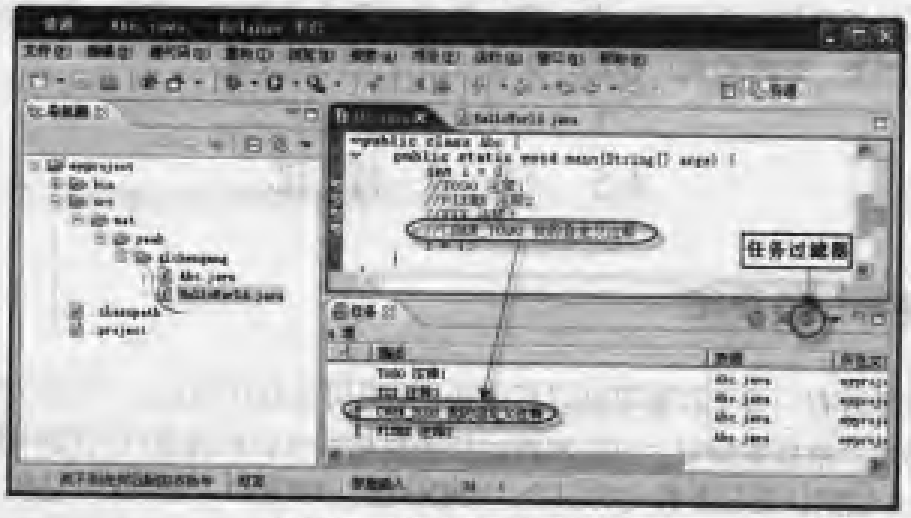


图 4.20　自定义注释的效果图

4.5.2　过滤任务标记

如果项目很大，任务视图中会有很多的任务标记，有些是我们不必关心的。要解决这个问题，一方面可以单击任务列表的表头来排序，另一方面也可以过滤掉一些任务。过滤任务标记的操作步骤如下：

（1）单击如图 4.20 所示小圈中标示的“任务过滤器”按钮。

（2）如图 4.21 所示，输入 CHEN_TODO，再单击“确定”按钮，这样任务视图中就只会显示出 CHEN_TODO 的任务标记。

图 4.21　“过滤器”窗口

本节只讲解了其中一种过滤方式，过滤组合方式很多，读者可按自己所需进行过滤。

4.6　在编程时随意查看 JDK、Eclipse 源代码

在 Eclipse 中阅读代码是很轻松的一件事。Eclipse 不仅提供了大纲、层次结构等总体性视图，还为代码跟踪提供了极其方便的操作。

4.6.1　查看 JDK 源代码

（1）如图 4.22 所示，我们想查看 println 方法在 JDK 中的源代码，操作步骤如下：按住 Ctrl 键不放，再单击该方法名。

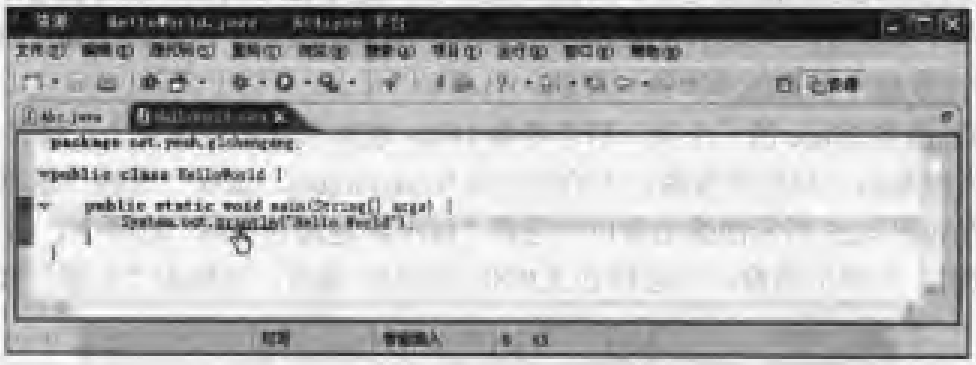


图 4.22　源代码跟踪

（2）由于是第一次查看 JDK 源代码，Eclipse 会提示要先设定源代码包的位置，如图 4.23 所示，单击“关联源代码（A）…”按钮。

（3）如图 4.24 所示，在弹出的窗口中输入 JDK 源代码包的路径 C:/jdk/src.zip，再单击“确定”按钮，代码编辑器中将显示相应的 JDK 源代码。

4.6.2　查看 Eclipse 的源代码

查看 Eclipse 源代码的初期设置要比 JDK 源代码麻烦一些。但阅读优秀的源代码是提高编程能力的最佳途径，所以麻烦一些也是值得的。而且从本实例还可以学习到如何设置对

库的引用，以及如何建立类路径变量。本节以查看一个 JFace 类的源代码为实例讲解。

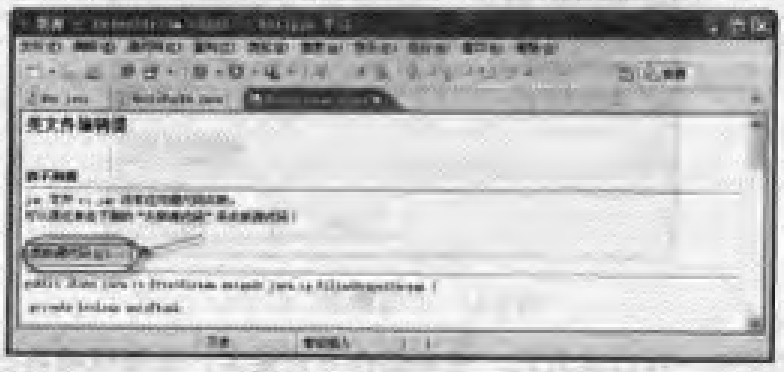


图 4.23　“找不到源”的提示

图 4.24　设定源代码位置的路径

第一步：将 JFace 包（或称 JFace 库）加入到 Eclipse 的构建路径上（也常称为库引用）。

构建路径的概念可以按照 JDK 的 classpath 来理解。网上经常会有人提问说：为什么完全照着例子写的 SWT 程序运行不了？import 语句报找不到类的错误？原因就是没有将 SWT/JFace 的包添加到构建路径中。Eclipse 是根据构建路径上的设置来寻找 SWT 包的，找不到 SWT 包，包中的类和组件也就没办法使用，自然就报错了，这和 JDK 下没有设置好 classpath 而导致程序报错的原因是一样的。

本实例要用到 JFace 的一个类，所以要将 JFace 包加入到构建路径上。操作步骤如下：

（1）右键单击“导航器视图”中的项目名称 myproject，选择“属性”选项。

（2）在如图 4.25 所示的弹出窗口中选择“Java 构建路径→库”选项卡，然后单击“添加变量”按钮。在弹出的窗口中选择 ECLIPSE_HOME 选项，再单击“扩展”按钮。

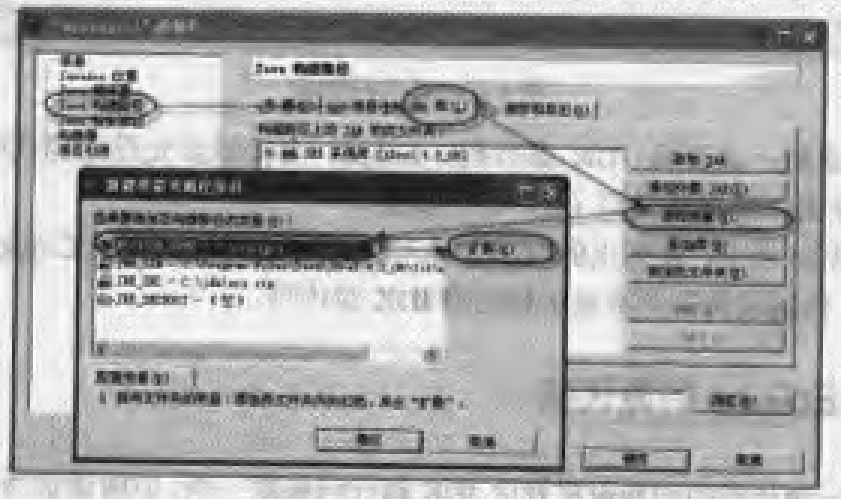


图 4.25　“项目属性”窗口

（3）在弹出的窗口中选择 plugins→org.eclipse.jface_3.0.0→jface.jar，单击“确定”按钮，返回“项目属性“窗口后，单击“确定”按钮，结束整个设置。

创建路径中的设置信息被保存在项目配置文件.classpath 中，双击可以打开该文件，如图 4.26 所示。如果觉得界面操作太繁琐，可以直接编辑此.classpath 文件来实现相关设置。

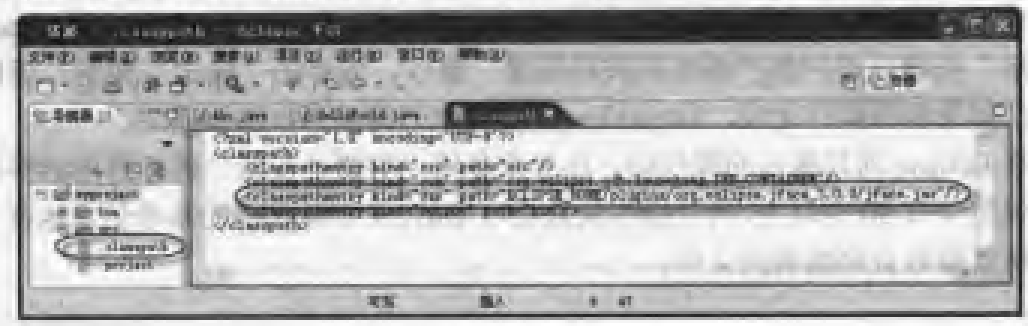


图 4.26　.classpath 文件

第二步：设置类路径变量、查看 JFace 的代码。

（1）编写一句用到了 JFace 包的程序，再次打开 HelloWorld.java 文件，在代码中加一行代码“TableViewer tv;”，这行代码是使用了 JFace 中的表格组件 TableViewer 类。按 Ctrl+Alt+O 键将所需要 import 的包导入。

（2）按住 Ctrl 键不放，再单击代码中的 TableViewer，与查看 JDK 源代码一样，需要设定一下 JFace 源代码的位置，不过操作方法和查看 JDK 源代码的不同，要麻烦一些。Eclipse 的源代码包主要在以下两个目录中：

```
C:\eclipse\plugins\org.eclipse.platform.source_3.0.1\src
C:\eclipse\plugins\org.eclipse.platform.source.win32.win32.x86_3.0.1\src
```

（4）单击“关联源代码”按钮，依次在先后弹出的窗口中各单击如下三个按钮：“变量→新建→文件夹”，如图 4.27 所示。

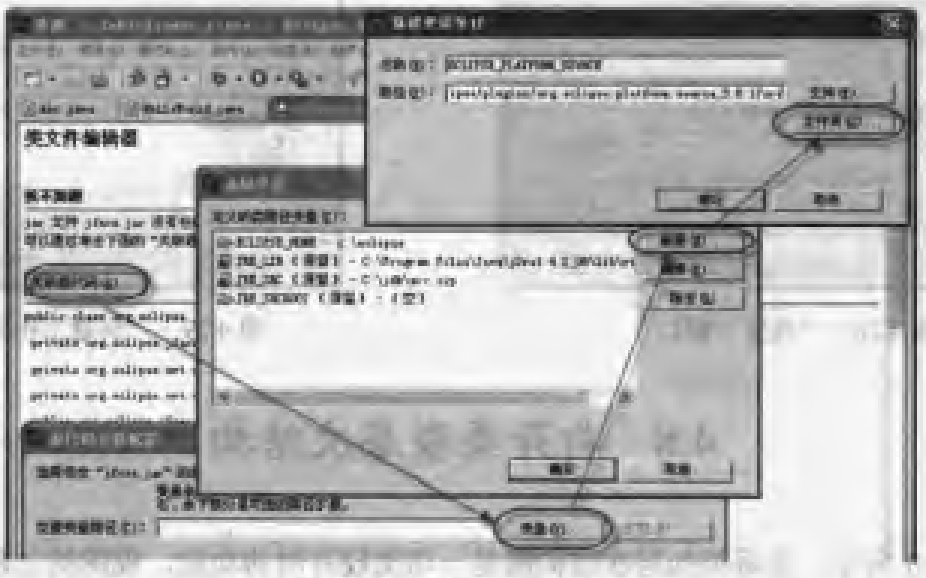


图 4.27　“新建变量”窗口

（5）然后选择目录 C:\eclipse\plugins\org.eclipse.platform.source_3.0.1\src，并输入名称

为 ECLIPSE_PLATFORM_SOURCE，单击“确定”按钮，返回“类路径变量”窗口。

（6）在类路径变量列表中选择 ECLIPSE_PLATFORM_SOURCE，单击“确定”按钮，返回“源代码关联配置”窗口，这时发现 ECLIPSE_PLATFORM_SOURCE 已经加入到文本框中了。

（7）单击“扩展”按钮，在打开的窗口中选择 org.eclipse.jface_3.0.0 下的 jfacesrc.zip，单击“确定”按钮，返回到“源代码关联配置”窗口，再单击“确定”按钮，返回到类文件编辑器。如果步骤操作无误，TableViewer 的源代码将显示在编辑器中。

要查看 Eclipse 的其他包的源代码，还要重复将“类路径变量”进行扩展的操作，这样的源代码包至少有 50 个以上，所以操作量是很可怕的。由于源代码的路径信息也在.classpath 文件中，所以最好直接编辑此文件。

4.7　在代码中搜索

Eclipse 的搜索功能也是开发中经常用到的功能，按 Ctrl+H 键，可以直接调出“搜索”窗口，如图 4.28 所示。搜索功能的使用方法比较容易，这里先介绍一下“工作集”概念。在这里，所谓“工作集”就是指希望搜索的文件的集合，这样的集合内容包括帮助文档、项目文件及配置文件等，如图 4.29 所示，就是通过复选框选择要搜索的项目文件。

如图 4.28 所示的选项卡，搜索分为四大类：文件搜索、Java 搜索、帮助搜索和插件搜索，当有符合条件的文件时，Eclipse 会将这些文件显示在“搜索”视图中。

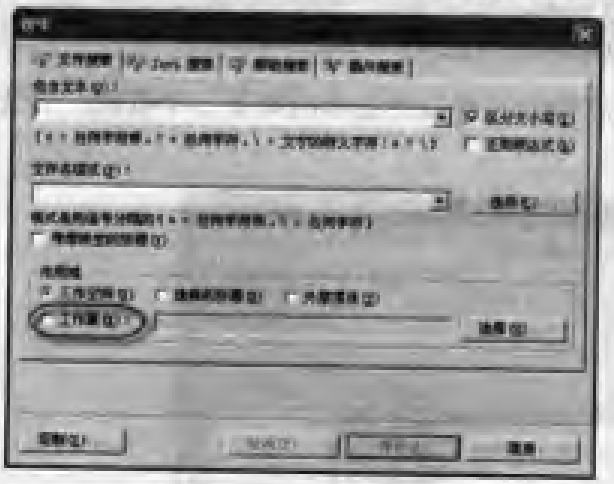


图 4.28　“搜索”窗口

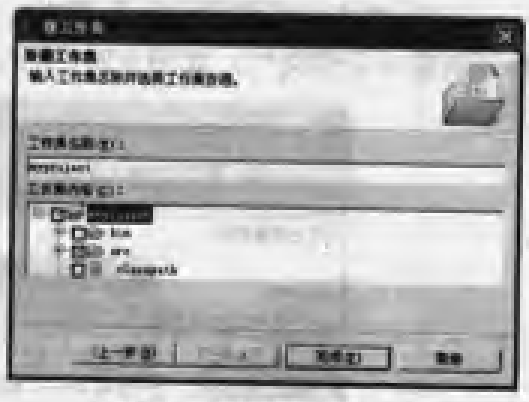


图 4.29　“新建工作集”窗口

4.8　打开类型层次结构

如图 4.30 所示，查看类的层次结构图是一项比较有用的功能，视图的上部为类的继承关系，下部为当前类的大纲，通过这些信息能更容易地看清某个类在整个类结构中的位置。

打开“层次结构”视图的操作方法是：先将光标移动到所要查看的类，再单击右键，

选择“打开类型层次结构”选项。

在“层次结构视图”上部，右键单击相应的类（如 StructuredViewer），选择“聚焦于“StructuredViewer””选项，也可以看到 StructuredViewer 类的继承关系。在“层次结构视图”上部，双击相应的类，则可以将其在代码编辑器中打开。

图 4.30　层次结构视图

4.9　调试器的使用

调试器是 Eclipse 的重要功能，同时熟练使用调试器也是程序员最基本的技能之一。为了方便讲解，改写 HelloWorld.java 程序如图 4.31 所示，并在编辑器左边显示行号。显示行号的方法是：单击主菜单“窗口→首选项”选项，再选择“Java”下的“编辑器”，在窗口右部选中是“显示行号”选项。

以下为调试时几个常见的功能。

（1）设定断点。在图 4.31 中箭头指向的位置，双击就可以为程序设置一个断点，Eclipse 以一个小圆点表示断点。再双击又可以将该断点取消。

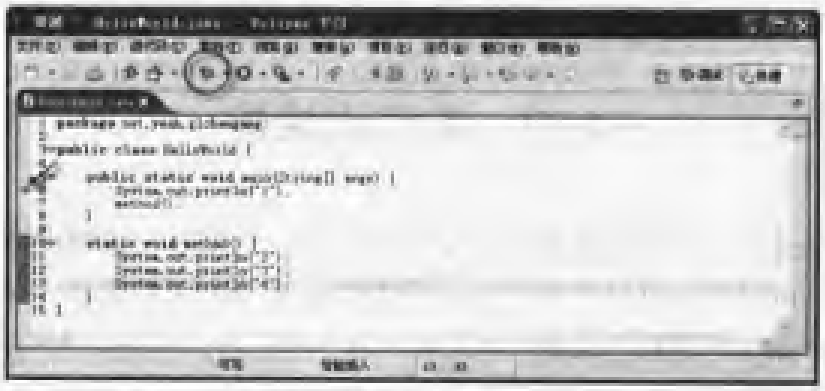


图 4.31　HelloWorld.java 程序

（2）调试程序与运行程序的方法相似，单击图 4.31 圆圈中的按钮，开始调试程序，Eclipse 将自动转到“调试透视图”。在此透视图中，最常用的是如图 4.32 所示的三个窗口，其中监视窗口可以监视代码中的变量、表达式的值的变化情况，也可以对断点做启用/失效、删除等操作。

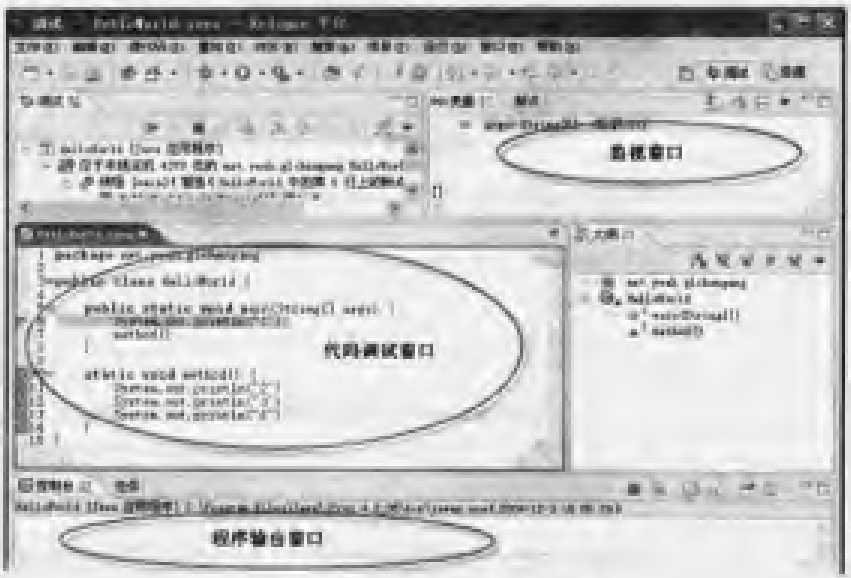


图 4.32　“调试”透视图

（3）单步跳入，快捷键 F5。运行当前行的代码，如果当前行调用了子方法，则会追踪进子方法中。如图 4.32 所示，当代码在第 6 行时，按 F5 键单步跳入，则会运行第 6 行的代码在控制台打印出“1”，然后当前行停在第 7 行。第 7 行是调用了一个子方法，按 F5 键单步跳入，追踪进此方法，这时当前行停留在第 11 行。

（4）单步跳过，快捷键 F6。如果当前行在第 7 行的子方法处，则按 F6 键单步跳过，将运行此子方法中所有程序，当前行停在第 8 行。“跳过”并非指不运行当前行。

（5）单步返回，快捷键 F7。如果当前行在第 11 行处，则按 F7 键单步返回，程序将运行完子方法剩下的代码，然后返回到主方法中，当前行停在第 8 行。

（6）继续，快捷键 F8。除非代码后面还有断点，否则程序将运行完毕。

（7）终止，主菜单“运行→终止”。程序将从当前行中断运行，当前行以后的所有代码也不再执行。

4.10　常用快捷键

在此给出一些最常用的快捷键和 Eclipse 的小技巧，它们对于编辑程序是极为有用的。要了解更多这方面的技巧，可以查看 Eclipse 的帮助。打开方法是选择“帮助→帮助内容→工作台用户指南→提示和技巧”选项。

- Alt + /：代码提示。
- Ctrl + /：注释/取消注释。

- Ctrl + D：删除光标所在行。
- Ctrl + K：将光标停留在变量上，按 Ctrl+K 键可查找下一个同样的变量。
- Shift + Ctrl + K：和 Ctrl + K 查找的方向相反。
- Shift + Ctrl + F：代码格式化。如果选择了代码，仅对所选代码格式化。
- Shift + Ctrl + O：快速地导入 import。
- Shift + Ctrl + X：将所选字符转为大写。
- Shift + Ctrl + Y：将所选字符转为小写。
- Ctrl + M：快速对当前视图最大化。
- Ctrl + O：打开类似大纲视图的小窗口。
- Ctrl + 鼠标单击：可以跟踪方法和类的源码。
- Ctrl + 鼠标停留：可以显示类和方法的源码。
- Alt + 左右方向键：跳到前一次/后一次的编辑位置，在代码跟踪时特别有用。
- 鼠标停留：可以显示类和方法的注释。
- 双击左括号（小括号，中括号，大括号），将选择括号内的所有内容。
- Ctrl + 1：光标停在某变量上，按 Ctrl + 1 键，可提供快速重构方案。选中若干行，按 Ctrl+1 键可将此段代码放入 for、while、if、do 或 try 等代码块中。

4.11　本 章 小 结

本章着重介绍了 Eclipse 开发环境的常用操作，这些内容足以让你马上进入开发状态。当然，Eclipse 的功能远不止这些，除了自带的众多视图之外，它还内嵌了 JUnit 测试框架、Ant。由于 Eclipse 开发环境的易用性，本书不把 Eclipse 开发环境的使用作为重点内容，如要了解更多内容，请参阅 Eclipse 的中文文档。

第 5 章　Eclipse 中 CVS 的使用

5.1　CVS 简介

CVS 是 Concurrent Versions System（并发版本系统）的简称。它是一个开放源代码的项目，是当前最流行的版本控制系统，目前绝大部分 Open Source 项目都使用它来做版本管理。如果你还习惯用微软的 VSS 来进行 Java 项目的版本管理，那么用上 Eclipse 后，建议还是转为使用 CVS。

CVS 采用客户机/服务器体系，代码、文档的各种版本都存储在服务器端，开发者首先从服务器上获得一份复制到本机，然后在此基础上进行开发。开发者可随时将新代码提交给服务器，当然也可以通过更新操作获得最新的代码，保持与其他开发者的一致。

Eclipse 本身内置了 CVS 客户端，只要再建立一个 CVS 服务器就可以使用这一功能强大的版本控制系统。CVS 的功能虽强大，但一般项目通常只用到其 20%的功能，所以只要了解最常用的操作就可以了，本章将以面向实际项目使用需要的方式来介绍 CVS。

5.2　CVS 服务器端的安装与配置

CVS 起源于 UNIX/Linux 平台，关于 UNIX/Linux 平台下的安装使用 CVS 服务器端的文章，在网上多如牛毛，本节不再重复。CVS 服务器在 Windows 平台的版本：cvsnt，它的安装有一定困难，本节将着重介绍 cvsnt 的安装。

首先到 cvsnt 主页 http://www.cvsnt.com/downloads/去下载它，下载方法参阅配书光盘的视频演示。目前的最新版本是 cvsnt 2.0.58d，它可以安装在 Windows NT/2000/XP/2003 上。

（1）运行安装程序，不必更改它的任何默认设置，连续单击 next 按钮，即可完成安装。cvsnt 默认安装在“C:\Program Files\cvsnt”目录下。

（2）通过 Windows 选择“开始→所有程序→CVSNT”选项，打开 cvsnt 的设置面板“Service control panel”，如图 5.1 所示，上面两个服务已经启动。

注意：如果提示找不到 rundll32.exe，则单击“浏览”按钮，指向“C:\windows\system32\rundll32.exe”并运行它。

（3）单击图 5.1 中的 Repositories 选项卡，再单击 Add 按钮，创建一个存放版本文件的目录“c:/cvsfile”，如图 5.2 所示，单击 OK 按钮后在弹出的两个窗口中回答“是”。

注意：建议在 Name 栏不要用自动生成的“/cvsfile”，而是改写成绝对路径“c:/cvsfile”，否则有可能在 cvsnt 安装后无法正常使用。

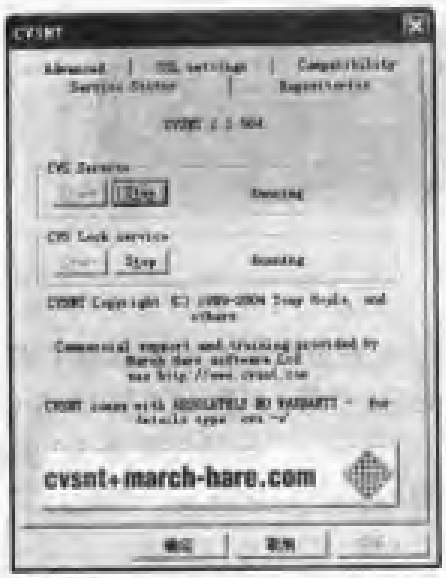


图 5.1　Service control panel 界面

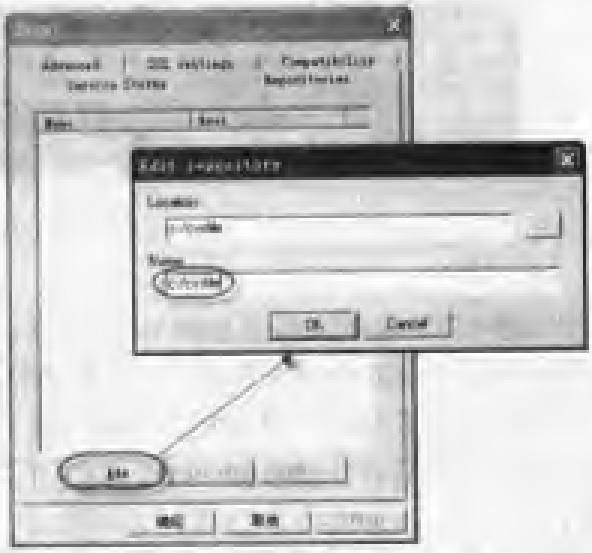


图 5.2　建立存放版本文件的目录

（4）转到“Compatibility”选项卡，将 3 个选项都选上，如图 5.3 所示。

注意：这一步非常重要，否则 CVS 2.0.58d 无法在 Eclipse 3.0.1 下正常使用，如果你使用 Eclipse 3.1M4 版，则此步设置可跳过。

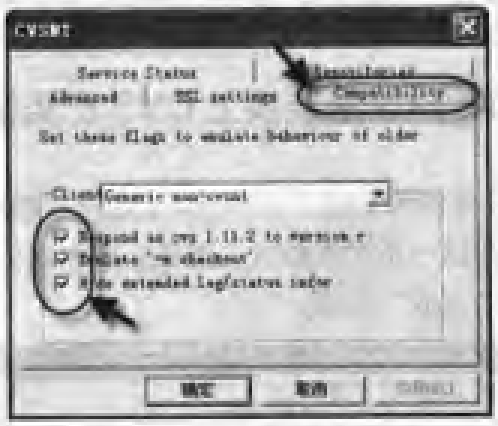


图 5.3　Compatibility 选项卡

（5）CVS 客户端要连接上服务器，还需要在服务器端分配一个用户名，并且用户名要求必须是 Windows 的登录用户。本书登录 Windows XP 的用户名为 giles，密码为空。进入 Windows 的命令行窗口，如图 5.4 所示，转到 C:\Program Files\cvsnt 目录下，输入命令 cvs -d c:\cvsfile passwd -a giles，按回车键输入 giles 在 CVS 上的登录密码（本书设置密码为 123456），此密码与 XP 可以不同。

注意：安装好 CVSNT 之后，也可以通过“控制面板”里的“用户账号”来创建一个新用户，这时该用户自动成为 CVS 用户，密码相同。

至此，CVS 服务器部分的设置已经全部完毕，客户端已可以使用 giles 为账号来登录 CVS。

图 5.4　命令行窗口

（6）其他

如果要将 CVS 的文件移植到另一台电脑上，或者原 cvs 服务器所在电脑重装了操作系统，这时就要涉及移植或恢复 CVS。以本书配书光盘的 cvsfile 目录（后面章节的 CVS 备份）为例，读者只需要将此目录复制到本地电脑（复制后也可以改名），然后按照图 5.2 的操作将它注册一下即可。由此可知，备份 CVS 上的文件只需将 cvsfile 目录复制一份就行了，非常简单。

如果读者对权限要求比较严格，那么就需要设置 CVS 用户的权限，CVS 的权限是和操作系统权限相关的（用户都是操作系统的用户），也就是说读者需要设置 Windows 文件目录的权限。

5.3　CVS 客户端的配置

5.3.1　配置 Eclipse 的客户端来连接上 CVS 服务器

打开 Eclipse 中的“CVS 资源库研究”透视图，然后在左部的“CVS 资源库”视图单击右键，在右键菜单中选择“新建→资源库位置”选项。在弹出的窗口输入 CVS 的相关信息，如图 5.5 所示，填完后单击“完成”按钮。

图 5.5　添加 CVS 资源库

注意：

（1）资源库路径用 CVS 服务器上的绝对路径，即图 5.2 中 Name 文本框的值。

（2）密码是上面用 cvs 命令行设置的 123456。但如果是通过“控制面板”里的“用户账号”来创建用户的，那么此密码即为该用户的 Windows 登录密码。

5.3.2　将项目提交到 CVS 服务器上

将项目提交到 CVS 服务器上的步骤如下：

（1）转到 Eclipse 的“资源”透视图，右键单击项目名 myproject，然后选择“小组→共享项目”选项，如图 5.6 所示。在弹出的“共享项目”窗口中，接受默认设置，直接单击“下一步”按钮。

图 5.6　“共享项目”窗口

（2）如图 5.7 所示，窗口提示输入模块名，接受默认选项，单击“下一步”按钮。

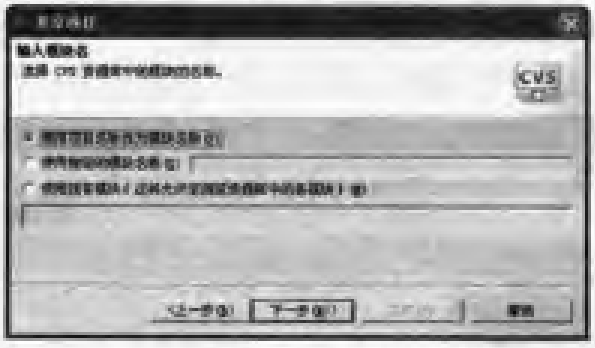


图 5.7　共享项目－输入模块名

（3）如图 5.8 所示，在窗口中选择要提交到 CVS 上的项目资源（即文件），不必作任何设置（默认为提交项目所有文件），直接单击“完成”按钮。

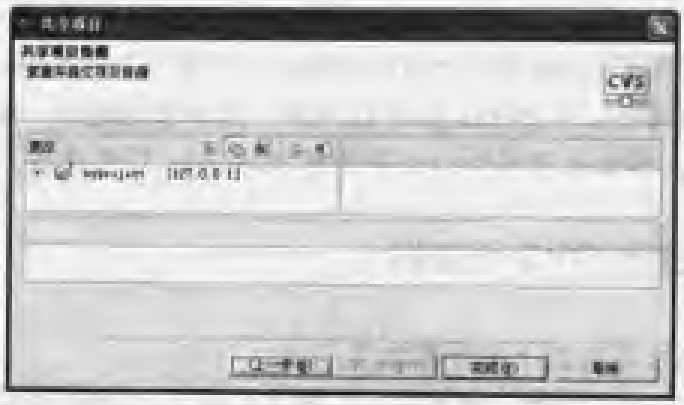


图 5.8　共享项目－项目资源

（4）如图 5.9 所示，连续弹出两个窗口，都单击“是”按钮。又弹出一个窗口要求输入提交注释，写上一些说明文字（也可以为空），单击“确定”按钮，Eclipse 开始将项目文件提交到 CVS 服务器上。

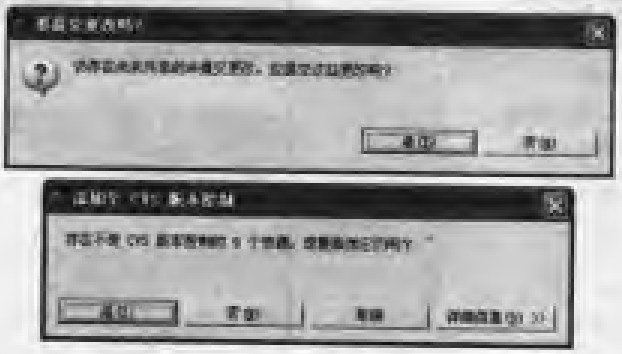


图 5.9　“资源提交”对话框

5.3.3　CVS 服务器上的项目导入到 Eclipse 中

如果新同事加入项目开发团队，这时他就需要将 CVS 服务器上的项目导入到自己的 Eclipse 中。为了方便模拟，再安装一个 Eclipse，只要安装目录不同，两个 Eclipse 同时运行也不会有冲突。将 CVS 服务器上的项目导入到 Eclipse 中的步骤如下：

（1）在主菜单选择“文件→导入”选项，弹出如图 5.10 所示的窗口。

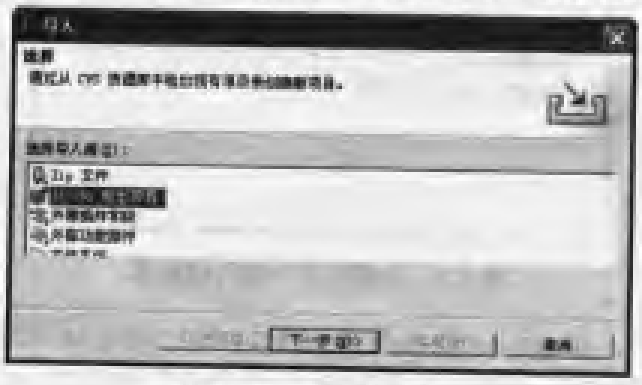


图 5.10　“导入”窗口

（2）选择“从 CVS 检出项目”，单击“下一步”按钮。弹出一个设定资源库的位置窗口，接受默认选项，直接单击“下一步”按钮，弹出如图 5.11 所示的窗口。

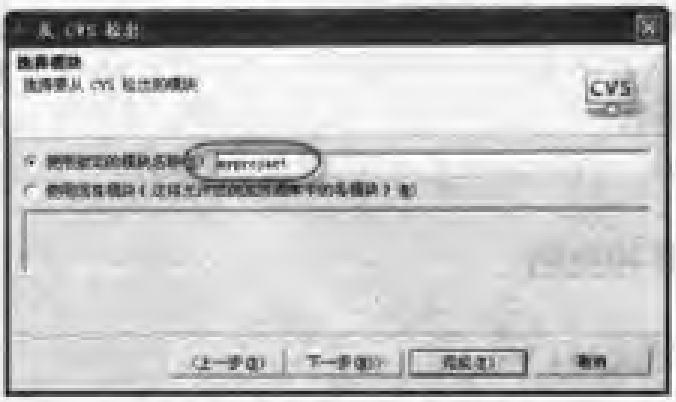


图 5.11　填写 CVS 上的模块名称

（3）在模块名称文本框输入 myproject，单击“下一步”按钮，弹出如图 5.12 所示的窗口。myproject 这个名称不是项目名称，而是项目共享到 CVS 上时所取的模块名（模块名的设置参见图 5.7），只不过通常将 CVS 模块名取成和项目名相同。

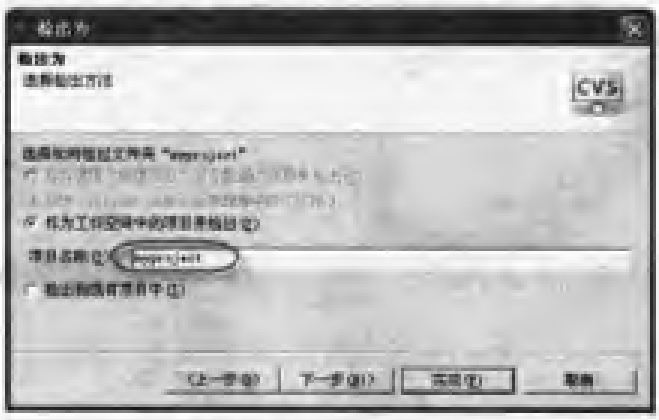


图 5.12　“检出为”窗口

（4）在项目名称文本框输入 myproject（可任意取名）。Eclipse 会以此名称生成一个项目，并将 CVS 的文件导出到此项目下。在该窗口中直接单击“完成”按钮，CVS 服务器上的最新文件开始导入到本机中。

5.4　文件提交与更新的方法

5.4.1　CVS 和 VSS 的不同之处

对于用过微软 VSS 的读者，在介绍 CVS 文件操作之前，有必要了解一下两者的不同之处：

（1）用 VSS 必须 check out（检出）文件才能修改此文件，而此时该文件将被锁定，其他人不能再修改（可以读），必须要等文件被上一个修改者 check in（提交）才行，也就

是说 VSS 同一文件一次只允许一个人修改。

（2）CVS 则使用完全不同的机制，多人可以同时修改同一文件。文件不会像 VSS 一样被锁定，所以也就没有修改文件之前要 check out(检出)的操作，CVS 的最基本操作是“提交文件到 CVS”和“从 CVS 更新文件到本机”。当然在 CVS 中如果多人同时修改了一个文件，在提交时会提示代码冲突，并提供一个窗口来合并这些修改，但这样的合并修改操作是很麻烦的，因此应尽可能地避免多人同时修改一个文件。

5.4.2　文件的提交和更新

1. 将修改的文件提交到 CVS

在 HelloWorld.java 中增加一条输出语句“System.out.println("第一次修改");”，然后保存。右键单击窗口右侧“导航器”视图中的 HelloWorld.java 文件，在弹出菜单中选择“小组→提交”选项，如图 5.13 所示。

图 5.13　提交界面

2. 从 CVS 更新文件到本机

删除掉 HelloWorld.java 中的语句“System.out.println("第一次修改");”然后保存，这时要将 CVS 服务器上的最新代码更新到本地，有以下两种方法：

（1）右键单击 HelloWorld.java 文件，在弹出的菜单中选择“替换为→HEAD 的最新内容”选项。

（2）右键单击 HelloWorld.java 文件，在弹出的菜单中选择“小组→与资源库同步”选项，在弹出的同步界面，如图 5.14 所示，单击“将当前更改从右边复制到左边”按钮，然后保存。

图 5.14　资源库同步界面

读者可能会问："右键单击 HelloWorld.java 文件，然后选择'小组→更新'这种更新方法，为什么不用呢？"

这种更新方法是将 CVS 的最新版更新到本机，当本机文件没有修改的时候才用这种方法。如果对 HelloWorld 作了修改，CVS 上的反而是旧内容了，这时再用这种更新方法则对本地的 HelloWorld.java 文件不会产生任何影响。

5.4.3　解决文件提交的冲突

在多人项目开发中，由于同时修改一个文件，一时就会产生提交冲突，本小节就来说明一下如何解决这种冲突。

在上面检出项目时，已经另外安装了一个 Eclipse，相当于多了一个开发者，原来的称为 Giles，新加入的叫 Tom。Giles 和 Tom 电脑上的 HelloWorld.java 的版本都是 1.2。这时 Giles 在文件中增加一条语句 "System.out.println("Giles 第 1 次修改");"，同样 Tom 也在文件中作了类似修改，如图 5.15 所示。

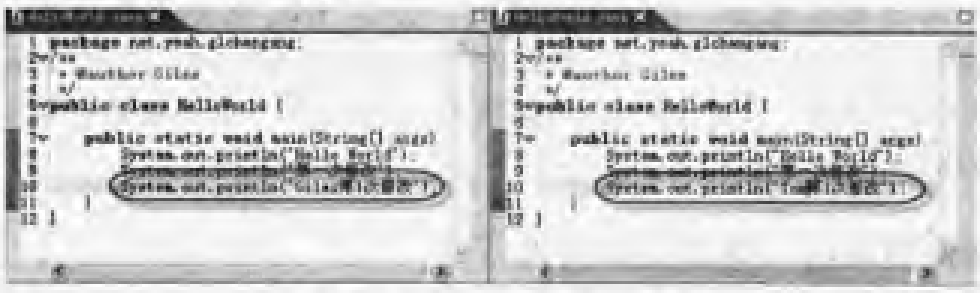


图 5.15　两开发者对代码的修改情况

Giles 先提交 HelloWorld.java 文件，这时版本变成了 1.3，由于 Giles 是第一个提交的，所以他不会遇到代码冲突。Tom 编程速度慢一些，所示后提交，这时就会和 Giles 先提交上去的代码产生冲突。因为如果用 Tom 的代码更新 CVS，则 Giles 的修改就会被冲洗掉，所以 Eclipse 此时就会报文件冲突，不允许直接提交。那么 Tom 该如何提交自己的修改呢？解

决办法如下：

右键单击“HelloWorld.java→小组→更新”选项，如图 5.16 所示。

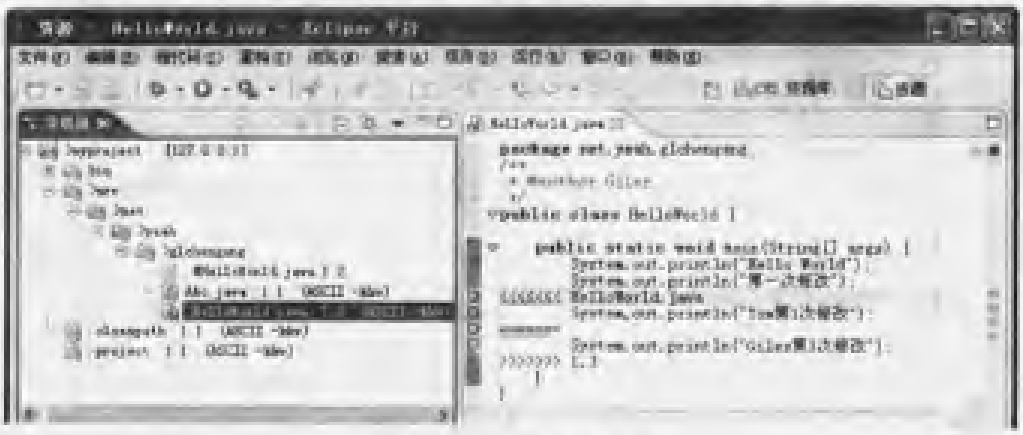


图 5.16　更新后的界面

此时 HelloWorld.java 的版本已经变成了 1.3，更新后的冲突也已标记出来。这时就可以根据代码中的标记来合并修改，修改完毕后再提交到 CVS，此时版本变成了 1.4，如图 5.17 所示。

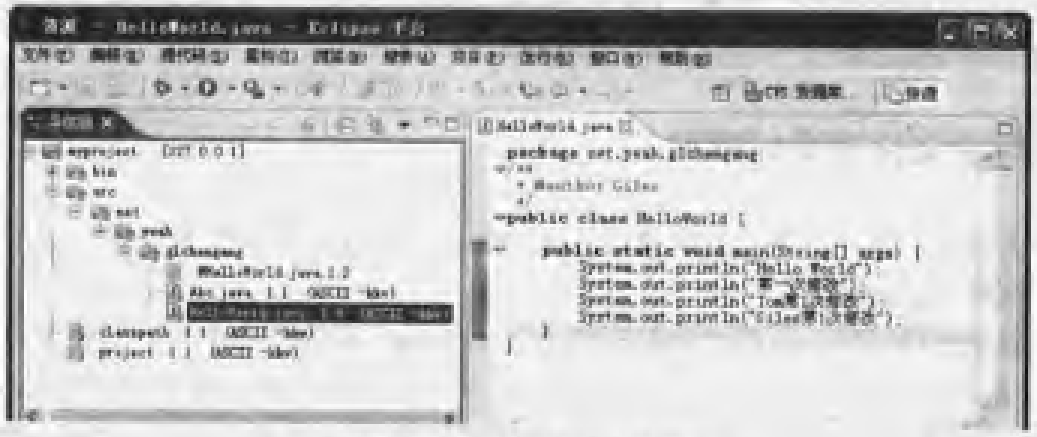


图 5.17　代码合并修改后的界面

Eclipse 考虑得很周到，在文件更新时，会将 Tom 的旧文件保存为一个称为“.#HelloWorld.java.1.2”的备份。这个备份文件不会被提交到 CVS，因为 Eclipse 中已经将这类文件作了忽略设置。忽略设置的界面如图 5.18 所示，在主菜单选择“窗口→首选项”选项可打开此窗口。

5.4.4　如何忽略掉不想提交的文件

假设当提交一个项目时出现如图 5.19 所示的情况，图中的 bin 目录包含的是编译后的 class 文件，没有必要将它提交到 CVS 上。我们可以右键单击它，然后在弹出的快捷菜单中选择“添加至.cvsignore”，这样 bin 目录就被忽略掉，不会提交到 CVS 上。

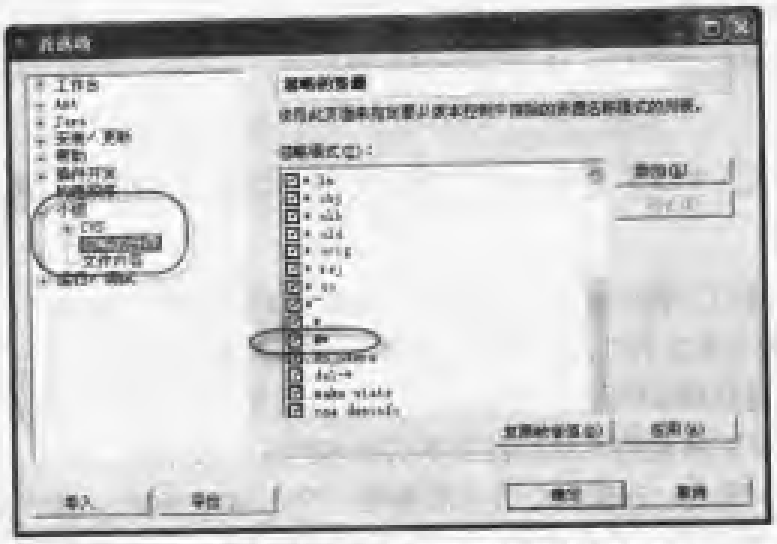


图 5.18　提交到 CVS 时被忽略的资源的设置界面

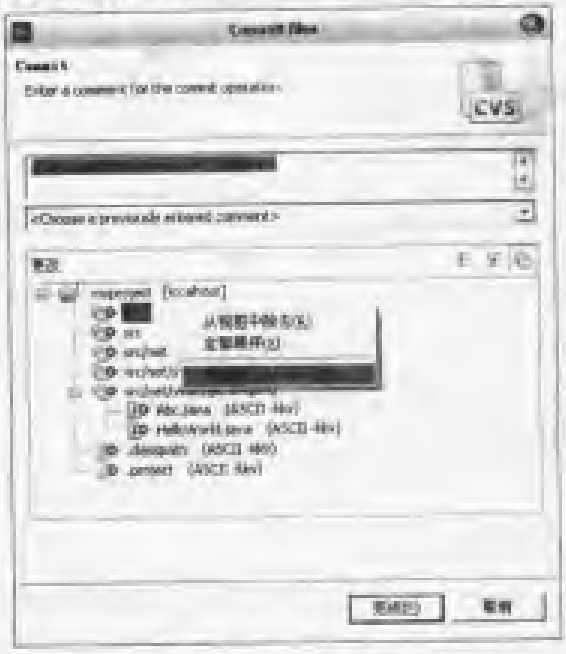


图 5.19　忽略提交

5.4.5　实践建议

（1）把项目做成模块目录（Java 包），各人负责自己的模块和目录，这样就不容易造成重复修改。

（2）做到每日提交，即每天下班之前都要将各自完成的代码提交到 CVS 上，并且提交的代码要求是完整可运行的代码，关键是要保证其他人更新你的代码之后不会引起错误。

（3）对于共用型的文件，要做到即时修改，即时提交。

（4）任何人在修改文件之前，先更新 CVS 的最新版到本机。

（5）项目团队的 Eclipse 等开发环境（安装目录、软件版本等）要保持一致。这样就可以将整个项目共享到 CVS，团队成员也可以从 CVS 上更新整个项目，而不必担心因为一

些项目配置文件不同导致更新项目后出错。另外，当新团队成员加入时，只要导入 CVS 上的整个项目，无须再配置项目，就可以即刻开发运行。

5.5　在 CVS 上为软件打包一个版本

在开发告一段落时，就要打包一个版本，打包版本的操作步骤如下：

（1）右键单击“项目名”，在弹出的快捷菜单中选择“小组→标记为版本”选项。此时如果有修改没有提交到 CVS 上，则会弹出下如图 5.20 所示的一个提示框，如果单击“确定”按钮，则未提交的修改就不会包含在本次打包的这一版本中。

（2）如图 5.21 所示，在窗口中为此版本起一个名称为 V001。由于此名称可能已存在于 CVS 中，所以下面还有一个复选框，如果选上该复选框，在打包时就会替换掉同名的老版本；如果没有选上，则在有同名版本时，就会弹出错误提示框。

图 5.20　确认未提交的更改

图 5.21　给版本起名

如果版本已经打包，但后来发现有一个文件有点问题，于是想把这一个修改的文件更新到上一个版本中，但另外一些修改的文件则不想更新到上一个版本。这时如果再通过右键单击项目名来打包，则项目所有修改的文件都会被更新到上一个版本中。

正确的操作步骤如下：

（1）将修改的文件提交到 CVS 上。

（2）右键单击该修改文件的“文件名”，在弹出的快捷菜单中选择“小组→标记为版本”选项。

（3）在图 5.21 中，还是起名 V001，并且要选择复选项，然后单击“确定”按钮。

此操作和打包项目版本惟一不同之处是：前者右键单击“项目名”，后者右键单击修改文件的“文件名”。

说明：CVS 打包版本并非是将项目文件再复制一份，如果去 c:\cvsfile 目录中查看提交的项目文件，就会发现项目文件只有一份，这一份文件中会记录着每一次的更改，所以 CVS 打包版本只是做了一个更改位置的标志。

5.6　将项目替换成 CVS 上的其他版本

接着 5.5 节内容，当打包成版本之后，如果要将此版本还原回项目中，则操作步骤如下：

右键单击“项目名”，然后在弹出的快捷菜单中选择“替换为→另一个版本或分支”选项，得到如图 5.22 所示窗口，选择版本 V001，单击“确定”按钮。

注意：此操作会将本地所有的修改都覆盖掉，所以在操作之前先把本地修改提交到 CVS 上保存。

图 5.22　CVS 的分支版本列表

5.7　修改旧版本的 BUG

如果已经发布了软件的 1.0 版，而现在正在开发 2.0 版，但用户在使用 1.0 版的时候发现了一些 BUG，这时就需要修改 1.0 版的这些 BUG，具体修改操作步骤如下：

（1）将 CVS 导出成另外一个项目。参见 5.3.3 小节，其他步骤都一样，仅在最后一步时，输入另外一个项目名 myproject2。这样分成两个项目后，修改旧版本和开发新版本就可以同时进行而不互相干扰。

（2）用 5.6 节的替换版本操作将 myproject2 项目替换成 V001 版本（假设 V001 就是出问题的 1.0 版）。如图 5.23 所示，在图中可以看到 myproject2 的后面跟着一个 V001 版本标志。

图 5.23　从 CVS 导入的 myproject2

（3）这时还不能在 myproject2 上修改，即使修改了也提交不上 CVS。因为还差一步，即需要为 V001 版创建一个分支，操作步骤如下：右键单击 myproject2 的项目名，然后选择“小组→分支”选项，在弹出如图 5.24 所示的窗口中输入分支名 V001_patch，并单击“确定”按钮。

说明：很容易把“分支”和“版本”看成类似的概念，但实际上“分支”是和 HEAD 的概念比较接近，“提交”操作只能将文件提交到 HEAD 或分支上。

（4）如图 5.25 所示，myproject2 的项目名后就有了一个分支名 V001_patch，表示该项目已经在这个分支下工作了。

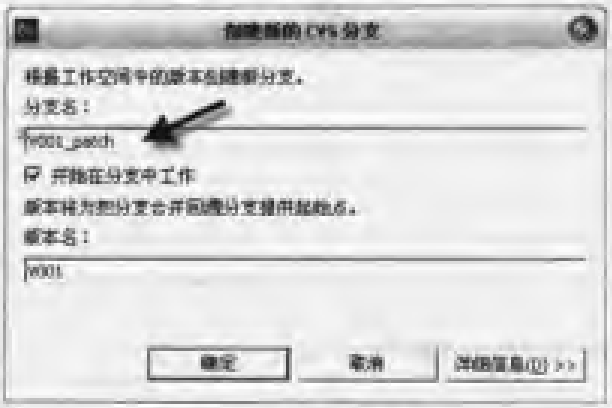


图 5.24　设定分支名

图 5.25　myproject2 在 V001_patch 分支下

（5）现在可以在 myproject2 项目中将 BUG 修改好，并提交到 CVS 上了。不过要说明的是，修改不是提交到 HEAD 上，而是提交到分支 V001_patch 上，所以也就不会影响到软件 2.0 版的开发。提交之后，文件尾部的版本号也起了变化，如图 5.26 所示。

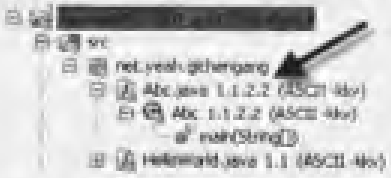


图 5.26　在分支下的程序版本号

（6）当修改完 BUG 后，就可以给软件再打一个版本 V100_1，并将修正了 BUG 后的版本发布给用户。

5.8　本章小结

本章介绍 CVS 的从服务器安装到客户端具体操作的整条开发线的使用流程，其中文件提交与更新是项目开发中使用最频繁的操作，本章详细讲述了具体的操作流程，并给出了由项目开发经验中总结出来的实践建议。通过本章的学习，读者已可以轻松地在项目中使用 CVS 来进行版本管理。

第 2 篇

SWT/JFace 篇

第 6 章　SWT 概述

第 7 章　SWT/JFace 的事件模型

第 8 章　SWT 常用组件

第 9 章　容器类

第 10 章　布局管理器

第 11 章　其他 SWT 组件

第 12 章　图像

第 13 章　SWT 的线程

第 14 章　表格（TableViewer 类）

第 15 章　树（TreeViewer 类）和列表（ListViewer 类）

第 16 章　对话框

第 6 章　SWT 概述

本章将把 SWT 和 AWT/SWING 作简单的比较，并以一个 HelloWorld 的 Java 应用程序作为起步，让读者可以快速建立对 SWT/JFace 的感性认识。本章中所有的例子都是以 Java 应用程序方式来写的，之所以如此，是因为 Java 应用程序代码简洁，并且可以独立运行，便于讲解和示范。当然，这些例子的代码方法同样适用于 Eclipse 的插件开发，SWT/JFace 在 Java 应用程序和 Eclipse 插件开发中的使用是没有太多区别的。

6.1　SWT 简介

2003 年，笔者对 SWT/JFace（Standard Widget Toolkit，SWT）还是仅有耳闻，知道在 AWT/SWING 之外，又有了一个新的图形 API 包，听说还很不错，当时书店里根本没有相关资料，只能在网上找到一些零星的文章来了解。

2004 年前，笔者还极少用 Java 来写 GUI（Graphical User Interfaces，图形用户界面）程序，主要的工作都是用 JSP 来写网页。用 Java 来开发大型的 GUI 程序实在是很困难的事，大多丑陋又笨重（慢），SUN 在 GUI 方向上的失败是公认的事实。失败关键之处在于 Java 的图形 API 包 AWT/SWING 在速度和外观上都不能让人满意，外观总是和同操作系统平台下的其他软件格格不入，对机器配置的需求也似乎永无止境。

2004 年初，笔者有幸参与到一个用 Eclipse 插件方式来开发的软件项目中，该软件使用到了 SWT/JFace，那界面实在是太酷太漂亮了，让人为之耳目一新，而且界面响应速度极快，这真的是用 Java 开发的吗？当时竟然有点不敢相信。

无疑，SWT/Jface 像一股清新的风吹入了 Java 的 GUI 开发领域，为这个沉闷的领域带来了勃勃生机。虽然 SUN 不接纳 SWT/JFace 作为 Java 中的一种图形 API 标准，但它仍然借着 Eclipse 的优异表现，以不可阻挡之势向前发展着。终于可以用 SWT 轻松地开发出高效率的 GUI 程序，并且拥有标准的 Windows 外观，Eclipse 软件就是基于 SWT/JFace 构建的，大家看看 Eclipse 3.0 就知道 SWT 有多么的棒。

如图 6.1 所示，为了方便开发 SWT 程序，在 SWT 基础上又创建了一个更易用、功能强大的图形包“JFace”。然而，JFace 并不能完全覆盖 SWT 的所有功能，所以编程时 SWT、JFace 都会被用到，但是一般来说，能用 JFace 的组件就最好不要用 SWT 的。

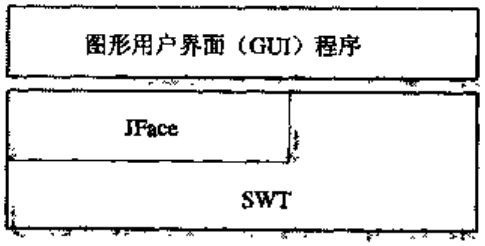


图 6.1　SWT、JFace、GUI 程序三者关系示意图

6.2　SWT 中的包

SWT 是 Eclipse 图形 API 的基础，本节将简单介绍 SWT 中所包含的子包。

1．org.eclipse.swt.widgets

最常用的组件基本都在该包中，如 Button、Text、Label、Combo 等。其中两个最重要的组件当数 Shell 和 Composite：Shell 相当于应用程序的主窗口；Composite 相当于 SWING 中的 Panel 对象，是容纳组件的容器。

2．org.eclipse.swt.layout

主要的界面布局方式在该包中。SWT 对组件的布局也采用了 AWT/SWING 中的 Layout 和 Layout Data 结合的方式。

3．org.eclipse.swt.custom

对一些基本图形组件的扩展在该包中，例如，其中的 CLabel 就是对标准 Label 组件的扩展，在 CLabel 上可以同时加入文字和图片。在该包中还有一个新的布局方式 StackLayout。

4．org.eclipse.swt.event

SWT 采用了和 AWT/SWING 一样的事件模型，在包中可以找到事件监听类和相应的事件对象。例如，鼠标事件监听器 MouseListener、MouseMoveListener 等，以及对应的事件对象 MouseEvent。

5．org.eclipse.swt.graphics

该包中包含针对图片、光标、字体或绘图 API。例如，可通过 Image 类调用系统中不同类型的图片文件。

6．org.eclipse.swt.ole.win32

对不同平台，SWT 有一些针对性的 API。例如，在 Windows 平台，可以通过该包很容易地调用 OLE 组件，这使得 SWT 程序也可以内嵌 IE 浏览器或 Word、Excel 等程序。

此外还有 org.eclipse.swt.dnd、org.eclipse.swt.printing、org.eclipse.swt.program、org.eclipse.swt.accessibility、org.eclipse.swt.browser、org.eclipse.swt.awt 等包，在此不一一介绍了。这些包一般很少用到，只需要稍微了解一下就行了，不必深究。

6.3　用 SWT Designer 写一个 Hello World

SWT Designer 是优秀的 SWT/JFace 开发辅助工具，本书中有关 SWT/JFace 的例子大部分都是使用它来生成代码后，再进行修改而成。当然，SWT Designer 并非是阅读和运行这些例子的必须条件。

本节将用 SWT Designer 来写出第一个基于 SWT 的 HelloWorld 程序，以此给读者演示在开发中是如何使用 SWT Designer 的。

6.3.1　使用向导建立一个 SWT/JFace Java 项目

（1）在主菜单选择“文件→新建→项目”选项，弹出如图 6.2 所示的窗口。

（2）在图 6.2 中选择 Designer 下的 SWT/JFace Java Project 选项，单击“下一步”按钮，弹出如图 6.3 所示的窗口。

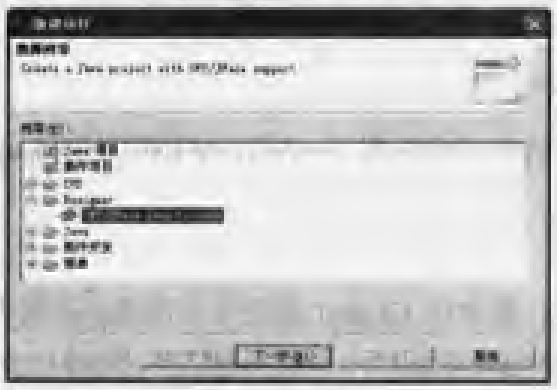


图 6.2　“新建项目”窗口

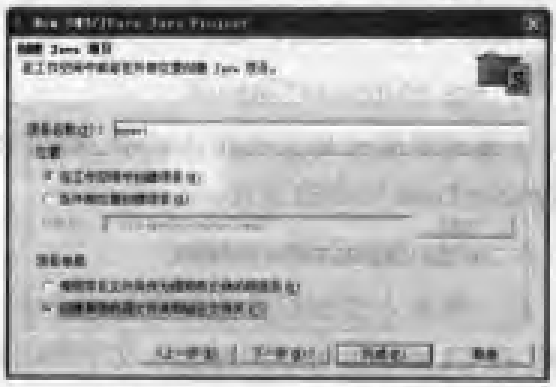


图 6.3　创建 Java 项目的窗口

（3）输入项目名称 myswt，项目布局选择“创建单独的源文件夹和输出文件夹”选项，单击“完成”按钮。这时如果打开 Java 透视图，可以看到多了一个名为 myswt 的项目，下方还排列着很多库引用，如图 6.4 所示的窗口。

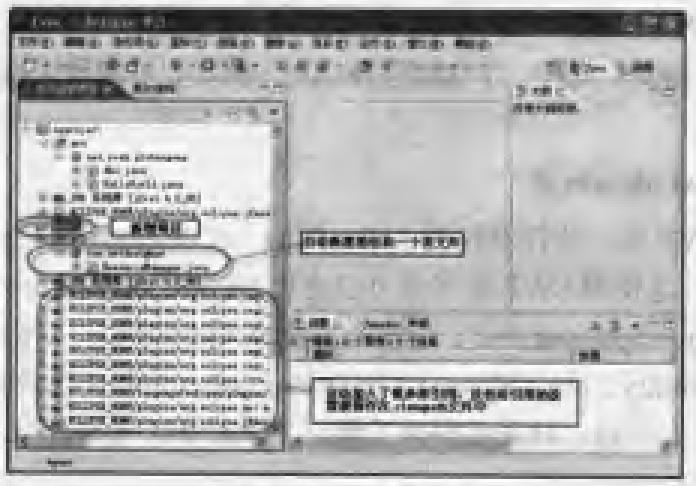


图 6.4　Java 透视图

注意：

（1）其实写 SWT 程序也不是一定要重新建立这样一个新的项目，原来老的 myproject 项目依然可以继续使用，但必须将 SWT、JFace 包及一些相关的包引用到 Java 构建路径中，手工一步步做这个工作太过于繁琐。有一种简单的方法：借助 SWT Designer 新建项目时保存在.classpath 文件中的库引用，将其复制粘贴到 myproject 的.classpath 中即可。

（2）当编写 Java 程序时，笔者认为 Java 透视图要比默认的“资源”透视图好用，主要是因为前者的包显示不是树状的，用起来较方便。但选择哪一种透视图，还要看各人的习惯和喜好。本书以后所讲的内容将统一使用 Java 透视图。

6.3.2　导入 SWT 的原生库

想要运行 Java 应用程序，必须将 SWT 的原生包导入到项目中，否则该项目在运行程序时会报异常“java.lang.UnsatisfiedLinkError: no swt-win32-3063 in java.library.path”，并弹出如图 6.5 所示的错误提示框。

导入 SWT 原生包的步骤如下：

（1）右键单击项目名 myswt，在弹出的菜单中选择“导入”选项，则会弹出如图 6.6 所示的窗口。

图 6.5　未导入 SWT 原生包时产生的错误提示框

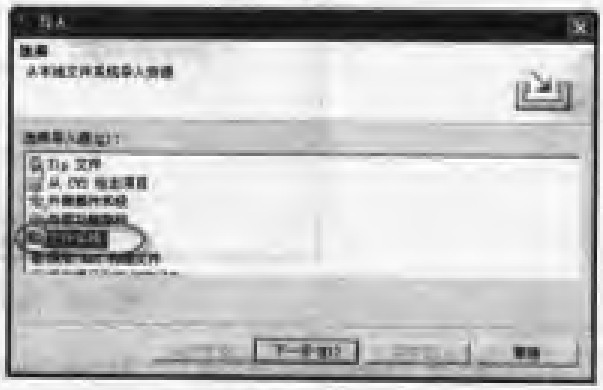


图 6.6　“导入”窗口

（2）选择“文件系统”选项后单击“下一步”按钮，转到如图 6.7 所示的窗口。

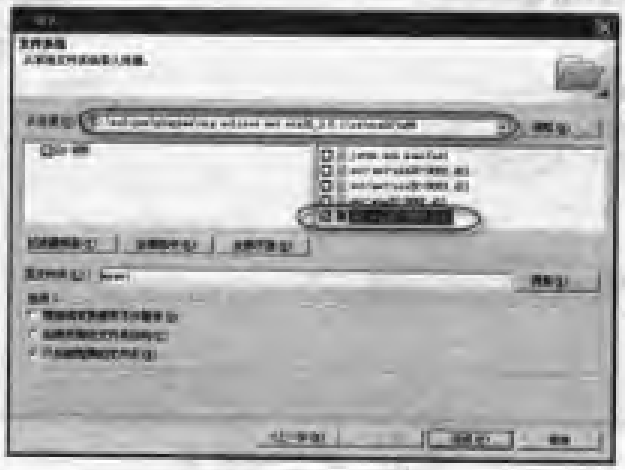


图 6.7　选择导入文件

（3）单击“浏览”按钮找到 SWT 原生库的路径（也可以直接输入路径文字），路径为“C:\eclipse\plugins\org.eclipse.swt.win32_3.0.1\os\win32\x86”，然后将“swt-win32-3063.dll”选上，单击“完成”按钮，导入 SWT 原生包的设置结束。

6.3.3　新建一个 SWT 类文件

参见 4.2 节所讲方法，新建一个类文件。

（1）在 Java 透视图的“包资源管理器”中，右键单击 com.swtdesigner 包，在弹出的菜单中选择“新建→其他”选项，弹出如图 6.8 所示的窗口。

图 6.8　选择新建的类型

（2）选择“Designer→SWT→Application Window”选项，单击“下一步”按钮，弹出如图 6.9 所示的窗口。

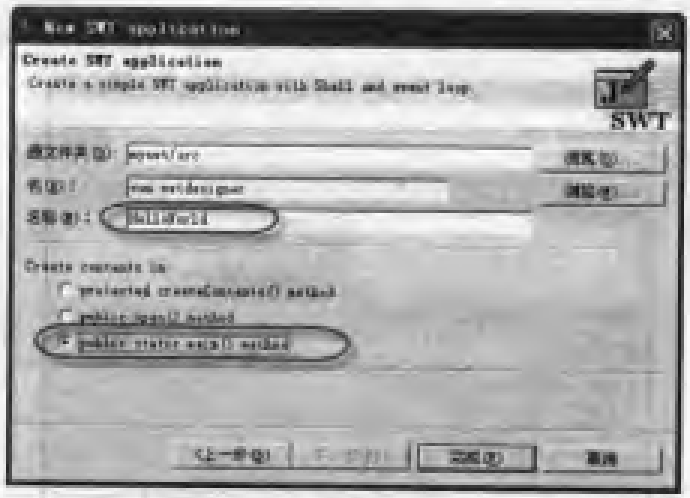


图 6.9　类文件的设置

（3）在类的名称中输入 HelloWorld，并选择“Create contents in（类代码的生成方式）”为“public static main() method”（第三项生成的代码结构最简单），单击“完成”按钮，Eclipse 将自动生成 HelloWorld.java 的代码，其代码如下（注释为笔者手工加入）：

```
package com.swtdesigner; //包名
import org.eclipse.swt.widgets.Display;//程序所用到的类都会用 import 标记在这里
```

```
import org.eclipse.swt.widgets.Shell;      //import 的快捷键 Shift+Ctrl+O
public class HelloWorld {   //一个标准的 Java 类 HelloWorld
    public static void main(String[] args) {
        //display 负责管理事件循环和控制 UI 线程和其他线程之间的通信
        final Display display = Display.getDefault();
        final Shell shell = new Shell();   // shell 是程序的主窗口
        shell.setSize(327, 253);   //设置主窗口的大小
        shell.setText("SWT Application");   //设置主窗口的标题
        shell.layout();   //shell 应用界面布置
        shell.open();   //打开 shell 主窗口
        while (!shell.isDisposed()) { //如果主窗口没有关闭，则一直循环
            if (!display.readAndDispatch())   //如果 display 不忙
                display.sleep(); //display 休眠
        }
    }
}
```

从这个代码可以看到，创建一个典型的 SWT 应用程序需要以下步骤：

- 创建一个 Display。
- 创建一个或多个 Shell。
- 设置 Shell 的布局（3.5 节将讲到布局的内容）。
- 创建 Shell 中的组件（本例还没有加入组件，只是一个空窗口）。
- 用 open()方法打开 Shell 窗口。
- 写一个事件转发循环。
- 销毁 Display。

6.3.4　在主窗口加入一个文本框组件

如果运行 HelloWorld.java 文件，它还仅是一个空荡荡的主窗口。我们利用 SWT Designer 将一个 SWT 的文本框组件加入到主窗口中，操作示意图如图 6.10 所示。

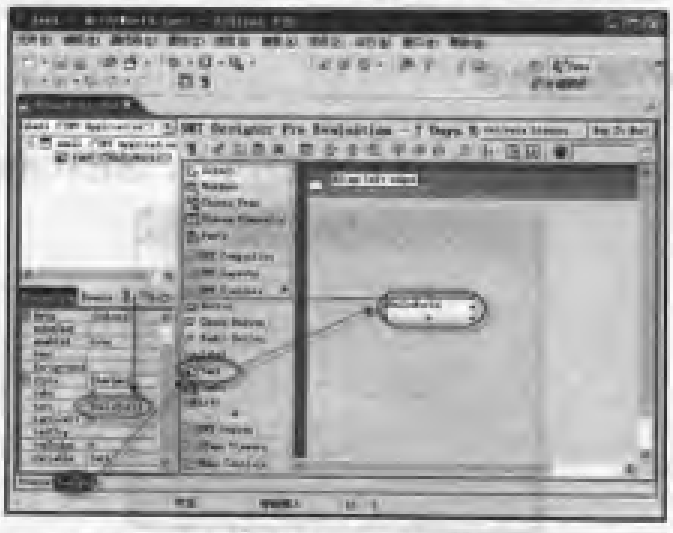


图 6.10　将文本框加入到主窗口的操作示意图

图中所示的操作步骤用文字描述如下：

（1）先将编辑器最大化，然后单击 Eclipse 的左下角的 Design 选项，则编辑器由代码视图变成设计视图。

（2）选择 SWT 组件栏中 SWT Controls 分类下的 Text 组件，然后在主窗口上单击，将 Text 框放入。注意这里不是通常地将组件拖入到窗口。

（3）转到属性窗口，在 text 项里输入 HelloWorld。单击 Eclipse 左下角的 Source 选项返回到编辑器的代码视图，其代码如下：

```
package com.swtdesigner;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Text;
public class HelloWorld {
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(327, 253);
        shell.setText("SWT Application");
        //----------------新插入的界面核心代码----------------------
        Text text = new Text(shell, SWT.BORDER); //新建一个 text 对象
text.setText("HelloWorld"); //给 text 文本框设置初始文字 HelloWorld
        text.setBounds(88, 94, 100, 25); //设置文本框的位置和大小(x 轴坐标,y 轴坐标,宽度,高度)
        //----------------END----------------------------------------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```

6.3.5　运行 HelloWorld.java

在主菜单选择 “运行→运行方式→Java 应用程序”，运行界面如图 6.11 所示。

图 6.11　HelloWorld 的第一次运行界面

以上的程序例子还是比较简单的，如图 6.12 所示，给出一个稍微标准些的界面，并给出了各类和界面之间的对应关系。注意，在 SWT 中 check 框（复选框）也是一种 Button。

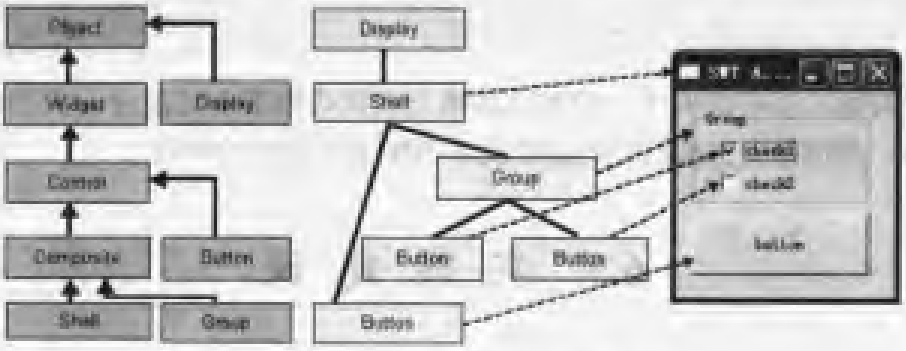


图 6.12　界面和类之间的对应关系图

其中 Display 和 Shell 的谱系图如图 6.13 所示，Group 和 Button 在 3.3 节有介绍。

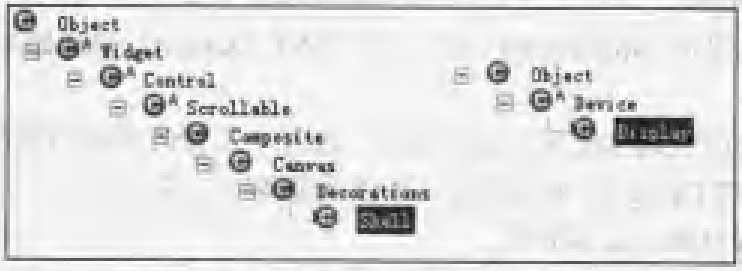


图 6.13　Display 和 Shell 的谱系图

6.4　关于 SWT/JFace 例程的说明

由于 SWT/JFace 应用程序例子的整体代码结构都基本一样，如下所示：

```
package com.swtdesigner;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Text;
public class HelloWorld {
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(327, 253);
        shell.setText("SWT Application");
        //----------新插入的界面核心代码---------
        ......
        //----------END----------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
```

```
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```

为了节省篇幅，以后的例子一般都会省略上面代码框架前后部分，只给出中间省略号处的核心代码，要想得到完整的代码请查阅本书配书光盘中的例程。

6.5 实 践 建 议

SWT Designer 还无法完成所有的界面设计工作，所以在界面开发中依然是以手工写代码为主，而且手写代码在某些时候比界面拖拉操作更快捷。以下是笔者在使用 SWT Designer 开发界面时的基本流程：

- 新建一个临时的 Application 文件，用 SWT Designer 快速做好开发所需要的部分界面。
- 将自动生成的代码移植到正式项目中，进行手工修改和代码精简。

另外，由于 SWT Designer 不很稳定，所以在使用时还应注意：

- 不要在界面中加入太多组件。
- 不要频繁地移动组件，或者删除又添加组件，否则很可能因为内存耗尽而死机。

6.6 本 章 小 结

本章主要介绍了 SWT 的一些基本知识，并且用 SWT Designer 开发出了本书的第一个 SWT 程序。通过本章的学习，读者对 SWT 有了一个初步的认识，并了解到如何用 SWT Designer 来开发 SWT 程序。

第 7 章　SWT/JFace 的事件模型

7.1　事件的 4 种写法

SWT 的事件模型是和 Java 标准的 AWT 基本一样的。在第 6 章的例子中，如何来实现文本框的事件响应呢？例如，鼠标双击文本框弹出一个对话框。下面将按照事件的四种写法来实现它。

7.1.1　匿名内部类写法

在原来的代码行“text = new Text(shell, SWT.BORDER);”之下插入如下语句：

```
//addMouseListener 加入鼠标事件的监听器
text.addMouseListener(new MouseAdapter() {
    public void mouseDoubleClick(MouseEvent e) {//鼠标双击事件的方法
        //打开一个信息框
        MessageDialog.openInformation (null,"","Hello World");
    }
});
```

new MouseAdapter()就是一个匿名内部类。我们建立了一个继承于 MouseAdapter 的类，但并没有给这个类命名，并且没有用通常的写法，而是直接在 text.addMouseListener 方法中写下了类的代码，这就是所谓的匿名内部类（更详尽的解释请参阅 Java 基础类书籍）。

使用匿名内部类来写事件代码简单方便，但也要注意它的一些缺点：

- 由于事件处理代码会随着组件一起分散在代码中的各个部分，不够集中，这样会导致代码阅读与维护上的不便。
- 各事件的处理全部由嵌套的程序块组成，视觉上会显示有些乱。如果事件处理代码很长，也会导致阅读与维护上的不便。
- 当工具栏、菜单栏等也需要处理相同的用户行为时，无法重用事件中的处理代码，导致了代码不够简洁。

7.1.2　命名内部类写法

事件代码使用命名内部类的方式，可以解决匿名内部类存在的问题：首先，事件处理代码都集中在一起，并且都具有有意义的名称，程序容易阅读与维护；另外，单个的事件处理程序也可以被工具栏、菜单栏等重用。实现代码如下：

```
public class HelloWorld {
    public static void main(String[] args) {
            ......
        Text text = new Text(shell, SWT.BORDER);
            //加入鼠标事件监听器，并用下面代码所定义的内部类生成一个对象
        text.addMouseListener(new MyMouseDoubleClick());
        ......
    }

        //定义一个名为 MyMouseDoubleClick 的内部类
    private static final class MyMouseDoubleClick extends MouseAdapter {
        public void mouseDoubleClick(MouseEvent e) {
            MessageDialog.openInformation(null, "", "Hello World");
        }
    }
}
```

7.1.3　外部类写法

这种写法和命名内部类有些相似，只不过是将 MyMouseDoubleClick 类从 HelloWorld.java 中拿出去，单独写成一个类文件。这种写法有和命名内部类一样的优点，但因为要单独写成一个文件，写起来会麻烦一些。实现代码如下：

```
//文件 1：  HelloWorld.java
public class HelloWorld {
    public static void main(String[] args) {
            ......
        Text text = new Text(shell, SWT.BORDER);
            //加入鼠标事件监听器，并用下面代码所定义的内部类生成一个对象
        text.addMouseListener(new MyMouseDoubleClick());
        ......
    }
}
```

```
//文件 2：MyMouseDoubleClick.java
public class MyMouseDoubleClick extends MouseAdapter {
    public void mouseDoubleClick(MouseEvent e) {
        MessageDialog.openInformation(null, "", "Hello World");
    }
}
```

7.1.4　实现监听接口的写法

将 HelloWorld 类实现 MouseListener 接口，这样类本身就成了一个监听器，使得加入监听器的代码可以更简洁。这种方式适合加入监听器的组件较多，并且要求监听器的事件处理代码可以被组件共用。这种方式还有一个要注意的地方：事件方法和其他方法混合在了

一起，容易引起误读，所以应该在事件方法前加入详细的注释说明。

实现 MouseListener 接口要写的事件方法多一些，当然没用的事件方法可以空实现。如果继承 MouseListener 接口的适配器 MouseAdapter，则只写需要的方法就行了。另外要注意：只有接口才能有多继承的特性，所以如果 HelloWorld 已经是某个类的子类，就只能用实现接口的方式，而不能继承接口的适配器了。

给出示例代码如下：

```
public class HelloWorld extends MouseAdapter{//或 implements MouseListener
    public static void main(String[] args) {
            ……
        Text text1 = new Text(shell, SWT.BORDER);
        Text text2 = new Text(shell, SWT.BORDER);
        text1.addMouseListener(this);
        text2.addMouseListener(this);
        ……
    }

    public void mouseDoubleClick(MouseEvent e) {
        MessageDialog.openInformation(null, "", "Hello World");
    }
}
```

7.1.5　总结

匿名内部类方式写起来要方便些，但不适合事件代码太长、太多的情况。从代码书写、阅读、维护以及程序的可扩展性角度来看，命名内部类写法最为值得推荐。外部类的写法主要是为了代码重用才考虑使用，如果包（package）外的类要用到此事件处理代码，这时外部类就派上用场了。而第四种写法，要求组件都可以共用事件代码时才能使用。

7.2　常用事件介绍

除了上例中用于响应鼠标事件的 addMouseListener，Eclipse 还有一些常用的监听器，它们在各组件中的使用方法相同（如果该组件支持此种事件），简单介绍如下：

- addSelectionListener：这个监听器最常用。
  - widgetSelected 方法：当组件被选择（单击鼠标、按回车键）时触发此方法的事件处理程序。
  - widgetDefaultSelected 方法：用于某些很少触发选择事件的组件，所以这个方法在实际开发中也很少用。例如，文本框回车事件、列表框双击事件等，就只能用 widgetDefaultSelected 方法，用 widgetSelected 方法无效。
- addKeyListener（按键）。
  - keyPressed 方法：当前焦点停在组件时，按下键盘任一键时触发。但对于某些

组件（如按钮 Button），按回车键无法执行此方法。

- keyReleased 方法：按键弹起时触发。

- addFocusListener（焦点）。
  - focusGained 方法：得到焦点时触发。
  - focusLost 方法：失去焦点时触发。
- addMouseListener（鼠标）。
  - mouseDown 方法：鼠标按下时触发。
  - mouseUp 方法：鼠标放开时触发。
  - mouseDoubleClick 方法：鼠标双击时触发。

以上几个就是常用的事件了，很少吧，SWT 的事件模型还是极容易掌握的。事实上除了 addSelectionListener 较常用之外，其他基本都很少用到。

7.3　在事件代码中如何访问类中的变量

7.3.1　访问类中变量的 3 种方法

在写事件代码的时候，常常需要引用主类中的变量，要访问这些变量是需要一些技巧的。

方法一：加 final 修饰符。

```
public class HelloWorld {
    public static void main(String[] args) {
            ……
            //将变量前加 final，否则在事件代码里不能引用
            final String str="陈刚";
            text.addMouseListener(new MouseAdapter() {
                public void mouseDoubleClick(MouseEvent e) {
                    System.out.println(str); //str 变量前要加 final
                }
            });
            ……
    }
}
```

方法二：将变量 str 变成类的实例变量。但这种方法扩大了 str 变量的有效范围。

```
public class HelloWorld {
    //由于引用它的代码是在静态方法内才加 static，否则不必要加 static
    static String str="陈刚";
    public static void main(String[] args) {
            ……
    }
}
```

方法三：将事件代码写成命名内部类，然后通过构造函数的参数来传入。这种方法较繁琐一些。

```
public class HelloWorld {
    public static void main(String[] args) {
        String str="陈刚";
        //通过构造函数参数将 str 值传入
        text.addMouseListener(new MyMouseDoubleClick(str));
    }
    //匿名内部类 MyMouseDoubleClick
    private static final class MyMouseDoubleClick extends MouseAdapter {
        private String string;//建一变量引用 str 的值
        public MyMouseDoubleClick(String str){ //通过构造函数参数接受 str 值
            this.string=str;
        }
        public void mouseDoubleClick(MouseEvent e) {
            System.out.println(string);
        }
    }
}
```

7.3.2　Java 中变量的称法和说明

本节中用到了一些 Java 变量方面的知识，在此一并附上。Java 中有三种容易混淆的变量：类变量、实例变量、局部变量，如下程序所示：

```
public class Variable {
    static int allClicks = 0; //类变量
    String str = "广西桂林"; //实例变量
    public void method() {
        int i = 0; //局部变量
    }
}
```

类变量的定义前加有 static，这表示它是一个静态的，因此类的多个实例共用一个类变量。实例变量定义在类的方法之外，一般处于类的起始位置，类的每一个实例都独自拥有一份实例变量的备份。局部变量的有效范围在程序块中，它的生命期仅限于此程序块内。

实例变量在有些书籍中也翻译成“域”或“成员变量”。在面向数据库的实体类（Hibernate 中也称 POJO，即简单原始的 Java 对象）中，被称之为“属性”或“字段”的变量，也是实例变量的一种。

使用变量的一般原则是，尽量使变量的有效范围最小化，即优先考虑用局部变量，其次是实例变量，最后才是类变量。

另外，还有一种常量的写法，它比类变量写法仅多了个 final，如下所示：

```
final static int ALL_CLICKS = 0; //常量
```

注意：ALL_CLICKS 全是大写的，这是常量的规范命名方式。这时 ALL_CLICKS 被 final 约束，它不能再被赋值了。

7.4 本 章 小 结

本章主要介绍了事件的四种写法，以及事件访问类中变量的方法，这些知识在 SWT 编程中经常要用到。但是，读者可以不必太执着于本章的内容，可以快速浏览后，进入第 8 章的学习，当用到时，再回过头来查阅。

第 8 章　SWT 常用组件

8.1　按钮、复选框、单选框（Button 类）

Button 是 SWT 中最常用的组件，而且使用也比较简单。Button 类的谱系图，如图 8.1 所示。

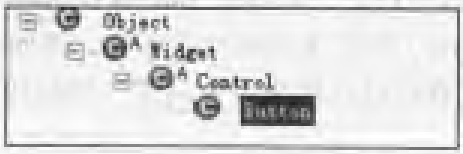


图 8.1　Button 类的谱系图

8.1.1　一个 Button 类的实例

这个例子是在窗口加一个按钮，单击该按钮出现一个提示框，且提示信息部分取自按钮上的文字，如图 8.2 所示。

图 8.2　按钮的运行效果图

以下代码只是核心部分，其他部分省略了（详情参阅 6.4 节）。

```
//------文件名:Button1.java------------
//事件代码里要访问 button，所以加一个 final
final Button button = new Button(shell, SWT.NONE);
button.addSelectionListener(new SelectionAdapter() { //加一个选择监听器
    public void widgetSelected(SelectionEvent e) {
        MessageDialog.openInformation(null, "", "你单击了" + button.getText() + "按钮");
    }
});
button.setBounds(50, 51, 100, 25); //设置按钮位置
button.setText("确定");//设置按钮上的文字
```

程序说明：

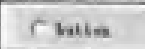
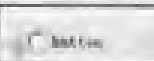
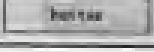
Button 类的构造方法是 new Button (Composite parent, int style)，它有两个参数：

- 第一参数是指 Button 创建在哪一个容器上。Composite 是最常用的容器，而 Shell 是 Composite 的子类，所以此参数也能接受 Shell 和任何继承自 Composite 的类。
- 第二个参数用来指定 Button 应用哪种（或几种）式样，SWT.NONE 是保持 Button 组件的默认式样。式样其实是一个常量，如 SWT.NONE 的值是 0，查看 SWT.class 的源代码可以发现很多这种常量。

8.1.2 Button 类的式样表

SWT 组件的式样是比较有特点的。和 new Button (Composite parent, int style)一样，大都在构造函数中使用式样（style）来定义组件外观形态。例如，Button 就可以通过式样的定义而成为复选框、单选框，或者通过式样来设置按钮文字的排放和按钮外观。表 8.1 给出了 Button 的全部式样。

表 8.1　Button的式样表

	SWT.PUSH	SWT.CHECK	SWT.RADIO	SWT.ARROW
仅SWT.NONE				
文字的式样				
SWT.CENTER（默认）	与SWT.NONE时一样			
SWT.LEFT				
SWT.RIGHT				
SWT.DOWN				
SWT.UP	与SWT.NONE时一样			
Button的外观				
SWT.BORDER（深陷型）				
SWT.FLAT（平面型）				

8.1.3 让 Button 应用多个式样

如果要生成一个文字靠左的、深陷型的复选框，只要用符号“|”将各个式样连起来即可，如下所示：new Button(shell, SWT. LEFT | SWT.BORDER | SWT.CHECK)。

8.2 组件的常用方法

SWT/JFace 中的每一个组件之间都有很多的同名的方法，很幸运，这些同名方法在各个

组件的作用和用法都是相同或相似的，这样就为我们节省了很多学习的精力。在此将一些常用的方法简述如下：

- setText (String string)
  说明：设置组件的标签文字。
  例子：button.setText("确定")。
- setToolTipText (String string)
  说明：设置鼠标停留在组件上时，出现的黄色提示条中的文字。
  例子：button.setToolTipText ("单击确定按钮，结束设置")。
- setBounds (int x, int y, int width, int height)
  说明：设置组件的坐标位置和大小，（x 轴坐标，y 轴坐标，按钮宽度，按钮长度）。
  例子：button.setBounds(46, 123, 99, 23)。
- setEnabled(boolean enabled)
  说明：设置组件是否可用。false 不可用，true（默认值）可用。
  例子：button. setEnabled(false)。
- setFont (Font font)
  说明：设置文字的字体。
  例子：button.setFont(ResourceManager.getFont("", 14, SWT.BOLD | SWT.ITALIC))，ResourceManager 是 SWT Designer 在建立 SWT 项目时自动生成的类。
- setSelection(boolean selected)
  说明：设置是否选上。true 为选上，false（默认值）为不选上。当 Button 为复选框或单选框时此方法才会有效。
  例子：button.setSelection(false)。
- setForeground (Color color)
  说明：设置前景色。
  例子：button.setForeground(Display.getCurrent().getSystemColor(SWT.COLOR_RED))
  或者 button.setForeground(ResourceManager.getColor(255, 255, 0))。
- setBackground (Color color)
  说明：设置背景色。
  例子：label.setBackground(Display.getCurrent().getSystemColor(SWT.COLOR_RED))。
- setAlignment (int alignment)
  说明：设置标签文字的对齐方式。
  例子：label.setAlignment(SWT.RIGHT)。
- setImage (Image image)
  说明：设置显示的图片。
  例子：button.setImage(new Image(display, "icons/selectAll.gif"))，把图片 selectAll.gif 显示在按钮上。

8.3　标签（Label 类）

1．Label 类的谱系图（如图 8.3 所示）

图 8.3　Label 类的谱系图

2．一个实例

Label 是 SWT 中最简单的界面组件，给出它的一个实例如下：

```
Label label = new Label(shell, SWT.NONE);
label.setBounds(38, 21, 100, 18);
label.setText("姓名：");
```

3．Label 类的式样列表（如表 8.2 所示）

表 8.2　Label类的式样表

仅SWT.NONE	Name：
文字式样（默认：SWT.LEFT靠左）	
SWT.CENTER（居中）	Name：
SWT.RIGHT（靠右）	Name：
SWT.WRAP（自动换行）	this is a good name：
外观形状	
SWT.BORDER（深陷型）	Name：
分栏符	
SWT.SEPARATOR（分栏符）	
SWT.SHADOW_IN	
SWT.SHADOW_NONE（不显示分栏符，一片灰白）	
SWT.HORIZONTAL（横放的分栏符）	

8.4　文本框（Text 类）

1. Text 类的谱系图（如图 8.4 所示）

图 8.4　Text 类的谱系图

2. 一个 Text 的实例

本实例将创建一个文本框，它有如下功能：

- 只能输入数字。
- 至少要输入一个值。
- 长度不能多于 10 个字符。

以上要求在实际项目中是很常见的，它的实现代码如下：

```
//--------文件名：Text1.java----------------
final Text text = new Text(shell, SWT.BORDER); //定义一个文本框
text.setBounds(18, 20, 153, 25); //文本框的定位
text.setTextLimit(10); //最多只能输入 10 个字符
//检验监听器，每输入一个字符前都会触发
text.addVerifyListener(new VerifyListener(){
    public void verifyText(VerifyEvent e) {
            //检查输入的字符(e.text)是否在 0123456789 这个字符串中，不在 indexOf 会返回-1
            boolean b = ("0123456789".indexOf(e.text) >= 0);
            e.doit =b; //doit 属性如果为 true，则字符允许输入，反之不允许
    }
});

final Button button = new Button(shell, SWT.NONE); //定义一个"确定"按钮
button.setText("确定");
button.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) { //按钮的单击事件
     //如果文本框没有输入，则出现一个警告窗，否则出现一个信息提示窗
     String str = text.getText();
     if (str == null || str.equals(""))
         MessageDialog.openWarning(shell, "", "请输入一个字符");
     else
         MessageDialog. openInformation (shell, "", "输入值通过检证");
    }
});
button.setBounds(21, 55, 100, 25);
```

注意到这次 MessageDialog 的第一个参数用了 shell 做参数，而不像以前一样用 null。两种不同的参数的区别在哪里呢？用 shell 时，则弹出提示框窗口时，Windows 的任务栏不会新增一个任务项；用 null 时，则 Windows 的任务栏会多出一个任务项，如图 8.5 所示。

图 8.5　两种 Windows 任务条的比较图

3. Text 类的式样列表（如表 8.3 所示）

表 8.3　Text类的式样列表

仅SWT.NONE	123
文字式样	
SWT.CENTER（居中）	123
SWT.LEFT（靠左）	和SWT.NONE一样
SWT.RIGHT（靠右）	123
SWT.MULTI（可以输入多行，须回车）	this is a good name
SWT.WRAP（无须回车，到行尾后自动换行）	this is a good name
SWT.PASSWORD（密码型）	***
SWT.READ_ONLY（只读），但setText方法仍可以设值	123
外观形状	
SWT.BORDER（深陷型）	123
SWT.V_SCROLL（带垂直滚动条）	this is a good name
SWT.H_SCROLL（带水平滚动条）	

4. 常用方法

- setEchoChar (char echo)
  说明：将输入的字符显示成参数 echo 表示的字符。
  例子：text.setEchoChar('*')，相当于 SWT.PASSWORD 式样。
- setTabs (int tabs)

说明：按 Tab 键时前进多少个空格的长度，默认值为 8 个。只有当 Text 的式样为 SWT.MULTI、SWT.V_SCROLL、SWT.H_SCROLL 时，此设置才会有效。

例子：setTabs (4)，按 Tab 键时前进 4 个空格的长度。

- setTopIndex (int index)

  说明：转到文本框的第 index 行，0 为第一行，此命令可以在文本框中进行快速行定位。

  例子：text.setTopIndex(0)，将当前行定位到首行上。

- setTextLimit (int limit)

  说明：设置最多只能输入 limit 个字符。

  例子：setTextLimit (10)，文本框最多只能输入 10 个字符。

8.5　下拉框（Combo 类）

1. Combo 类的谱系图（如图 8.6 所示）

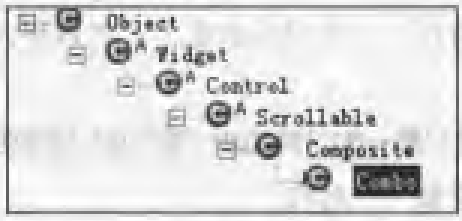


图 8.6　Combo 类的谱系图

2. 一个 Combo 类的实例

实例的界面如图 8.7 所示。

图 8.7　运行效果图

实例功能：下拉框最初没有值，单击“设置”按钮后将一组数据加入。单击“取值”按钮，弹出一个提示窗口，提示文字由下拉框的当前值组成。具体代码如下：

```
//————文件名：Combo1.java————————
final Combo combo = new Combo(shell, SWT.READ_ONLY); //定义一个只读的下拉框
combo.setBounds(16, 11, 100, 25);
//设值按钮
final Button button1 = new Button(shell, SWT.NONE);
button1.setBounds(17, 65, 100, 25);
button1.setText("设值");
button1.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
```

```
                combo.removeAll(); //先清空 combo，以防“设值”按钮多次按下时出现 BUG
                for (int i = 1; i <= 10; i++) //循环，赋值
                    combo.add("第" + i + "个字符串"); //在 combo 中显示的字符串
                combo.select(0); //设置第一项为当前项
            }
});

//取值按钮
final Button button2 = new Button(shell, SWT.NONE);
button2.setBounds(136, 66, 100, 25);
button2.setText("取值");
button2.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        // combo.getText()得到 combo 中当前显示的字符串
        MessageDialog.openInformation(shell, null, combo.getText());
    }
});
```

程序说明：

在这个例子中最关键的是设置值部分的代码，它是用 add()方法。还有另一种 Combo 类设置值的方法，是将所有值存入一个字符串数组，然后将此数组作为参数，批量加入到 Combo 中。语句如下：

```
combo.setItems(new String[] { "第一项", "第二项", "第三项", "第四项" });
```

3．让 Combo 的各项和对象一一对应

Combo 中的项只能是字符串，但在实际项目开发中会有很多这样的需求：将 Combo 中的各项和一些对象（不仅仅是字符串）一一对应起来。这些对象可以是任何类，这样就可以根据选择的项来得到相对应的对象。

要实现这样的功能，使用 setData 方法就行了。如果基于上一个例子，只需要修改设值和取值的事件代码。代码如下：

```
//--------文件名：Combo2.java----------
final Combo combo = new Combo(shell, SWT.READ_ONLY); //定义一个只读的下拉框
combo.setBounds(16, 11, 100, 25);
//设值按钮
final Button button1 = new Button(shell, SWT.NONE);
button1.setBounds(17, 65, 100, 25);
button1.setText("设值");
button1.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        combo.removeAll();
        for (int i = 1; i <= 10; i++) {
            combo.add("第" + i + "个字符串");
            combo.setData("" + (i - 1), new Integer(i));
        }
        combo.select(0);
    }
```

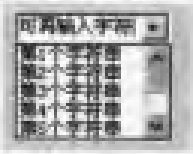
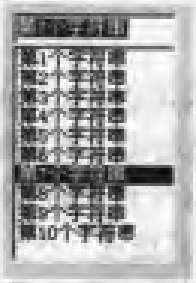
```
});

//取值按钮
final Button button2 = new Button(shell, SWT.NONE);
button2.setBounds(136, 66, 100, 25);
button2.setText("取值");
button2.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        //根据 SelectionIndex 值转化成字符串后得到对应的 key
        String key = "" + combo.getSelectionIndex();
        //根据 key 值得到 Object（对象），注意还要用(Integer)进行类型转换
        Integer selectObj = (Integer) combo.getData(key);
        MessageDialog.openInformation(shell, null, "选择的 Integer 对象的值是" +
selectObj.toSring());
    }
});
```

程序说明：

setData 方法的格式是：setData(key,Object)，它将对象（Object）对应一个键值（key），然后附着在 comb 上。键值可以是任意的字符串，本例之所以用"" + (i − 1)，是因为它的值正好和 combo 的 SelectionIndex 值（当前项的序号）相同，这样在取值时就方便了。另外，SelectionIndex 的起始值是 0，所以要“i-1”（i 的循环初始值是 1）。

4．Combo 类的式样列表（如表 8.4 所示）

表 8.4　Combo类的式样列表

式　样	例　图
仅SWT.NONE	
SWT.READ_ONLY（只读）	
SWT.SIMPLE （无须单击下拉框，列表一直都显示着）	

5．常用方法

Combo 的常用方法如下，还有其他常用方法参见 8.2 节。

- add (String string)

  说明：在 Combo 中增加一项。

  例子：combo.add("新项")。

- add (String string, int index)
  说明：在 Combo 中插入一项。
  例子：combo.add("新项",3)，在第 3 项的后面插入一项。
- deselectAll ()
  说明：使 Combo 组件中的当前选择项置空，即 Combo 没有选择任何一项。
- setItems(String[]　items)
  说明：将数组中的各项依次加入到 Combo 中，相当于执行多种 add 方法。
  例子：combo.setItems(new String[] { "第一项", "第二项", "第三项", "第四项" });。
- select(int index)
  说明：将 Combo 的第 index+1 项设置为当前选择项。
  例子：combo.select(0); 使首项为当前选择项。
- setText(String str)
  说明：此方法常用于设置当前选择项，str 是下拉框中的某一项。这个方法和上一个 select 方法虽然都能设置当前选择项，但效果却略有不同：setText 没有选择的背景阴影，select 方法则有。
  例子：combo.setText("中国");将下拉框的“中国”项设置为当前选择项。

8.6　列表框（List 类）

1．List 类的谱系图（如图 8.8 所示）

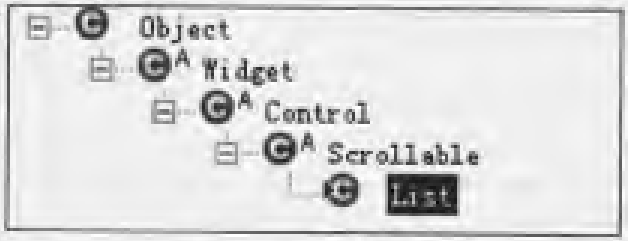


图 8.8　List 类的谱系图

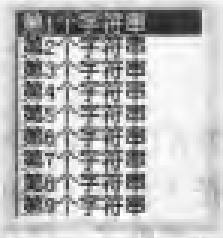
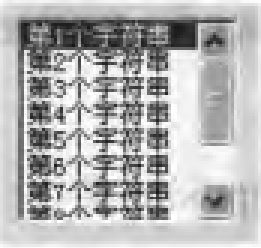
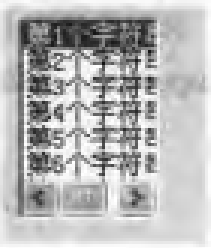
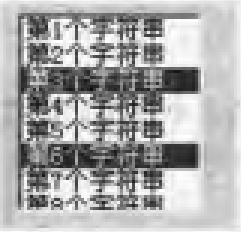
List 的用法与 Combo 基本是一样的，Combo 的两个例子只要将 Text 的定义部分改为 List 的定义语句，再稍微修改，即可成为 List 类的例子。

在 List 中设值的方法和 Combo 是一样的，但取值有所不同，这是因为 List 可选择多项，而 Combo 只能选一项；Combo 的 getText 可以得到当前选项的字符串，而 List 由于可以有多选，所以就没有 getText 方法了。

List 的取值是用 getSelection()方法，它返回的是一个所有选项组成的 String 数组。

2. List 类的式样表（如表 8.5 所示）

表 8.5　List类的式样表

仅SWT.NONE		SWT.BORDER （带边框）	
SWT.V_SCROLL （带垂直滚动条）		SWT.H_SCROLL （带水平滚动条）	
SWT.MULTI （可多选）			

8.7　本 章 小 结

本章讲解了 SWT 的常用组件，这些组件的使用相对比较简单，但在项目开发时却是最常用的。其中 Combo 组件让下拉框中的值与一组对象一一对应起来，这种使用方式应该掌握，在开发中很有用。

第9章 容 器 类

虽然本书在这之前的例子中的组件都是构建在 Shell 上，但 Composite 才是最常用的容器类。如果把主窗口（Shell）比喻成桌子，容器类就是桌子的抽屉，而抽屉里的物品就是一个个的界面组件了。组件是构建在容器类中的，这样就可以通过容器来对组件进行集体操作，例如，容器在界面上移动，其组件也会跟着移动；容器隐藏，其组件也会被隐藏；容器销毁（dispose），其组件也会被销毁。

9.1 面板（Composite 类）

1. Composite 类谱系

之前的例子中，用的 Shell 其实是 Composite 的一个子类，如图 9.1 所示。

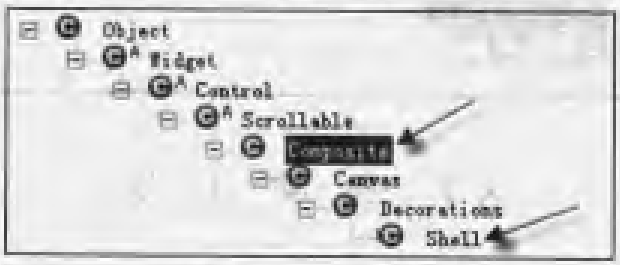


图 9.1 Composite 的谱系图

2. Composite 的用法

格式：Composite (Composite parent, int style)

用法：Composite composite = new Composite(shell, SWT.NONE);

说明：这里第一个参数还是用了 Shell 类，因为 Shell 类属于 Composite 的子类，所以 shell 也可以当做 Composite 类型来用。Composite 的式样一般都是用 SWT.NONE，这时 Composite 在界面是不显示出来的，只暗地里发挥着容器的作用。当然也可以用 SWT.BORDER 式样让它形成凹陷效果。

3. 常用方法

- composite.getLayout() 得到布局管理器。
- composite.getLayoutData() 得到本身的布局数据。
- composite.getParent() 得到容纳 composite 的父容器。
- composite.getShell() 得到容纳 composite 的 Shell。
- composite.layout() 将 composite 的组件重新布局，相当于刷新功能。

9.2 分组框（Group 类）

1. Group 类的谱系

如图 9.2 所示，可以发现 Group 其实是 Composite 的子类。

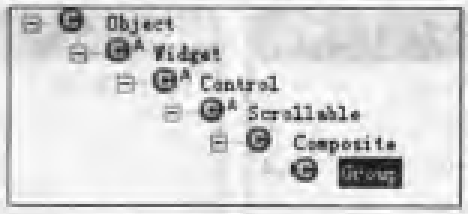


图 9.2 Group 类的谱系图

2. Group 的实际界面

Group 常用于集合一些同类界面组件，图 9.3 中就有三个 Group：有效更新、更新策略和代理设置。

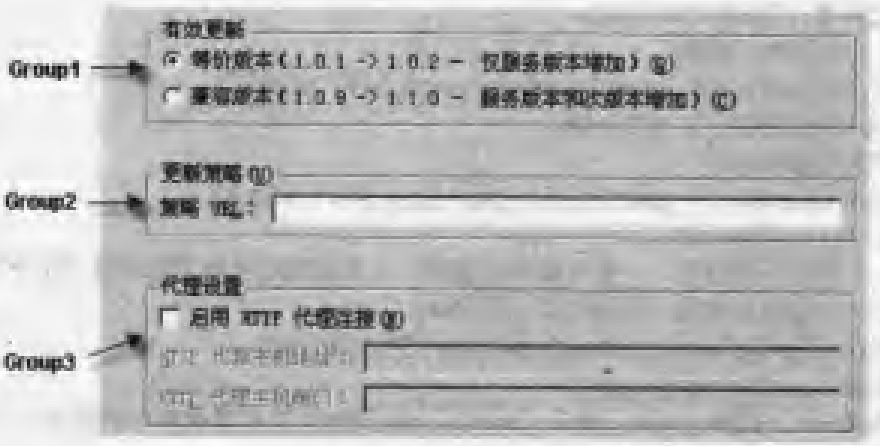


图 9.3 Group 示意图

3. Group 的用法

Group 是 Composite 的子类，所以 Group 和 Composite 是基本一样的。主要区别是 Group 显示有一个方框，且方框线上可以显示一串说明文字。显示此文字的方法是：

```
group.setText("代理设置");
```

9.3 选项卡（TabFolder 类和 TabItem 类）

1. TabFolder 类和 TabItem 类的谱系图（如图 9.4 所示）

2. TabFolder 类、TabItem 类的实例

如图 9.5 所示，这个实例在窗口中建立一个选项卡（TabFolder），它有两个选项页

（TabItem），图中分别为两个选项页的显示界面。

图 9.4　TabFolder 类和 TabItem 类的谱系图

图 9.5　TabFolder 类和 TabItem 类的实例界面

实例的代码如下：

```
//---------文件名：TabFolder1.java----------
//定义一个 TabFolder 容器
final TabFolder tabFolder = new TabFolder(shell, SWT.NONE);
tabFolder.setBounds(6, 5, 168, 118);
//第一个选项页 tabItem1
final TabItem tabItem_1 = new TabItem(tabFolder, SWT.NONE);
tabItem_1.setText("Item1");
{    //加一个空括号的目的：一是代码层次清晰；二是使程序块的局部变量互不干扰
     //定义一个 Group 容器，它建立在 tabFolder 上
     final Group group = new Group(tabFolder, SWT.NONE);
     group.setText("Radio Group");
     //让 tabItem_1 控制 group。单击 tabItem_1 显示 group 容器及其中的组件
     tabItem_1.setControl(group);

     //定义一个 Radio 按钮，注意它建立在 group 上
     final Button button_1 = new Button(group, SWT.RADIO);
     button_1.setBounds(25, 21, 100, 25);
     button_1.setText("radio1");

     //定义的另一个 Radio 按钮
     final Button button_2 = new Button(group, SWT.RADIO);
     button_2.setBounds(24, 50, 100, 25);
     button_2.setText("radio2");
}

//第二个选项页 tabItem2
final TabItem tabItem_2 = new TabItem(tabFolder, SWT.NONE);
tabItem_2.setText("Item2");
{
```

```
    //定义一个 Composite 容器，同样建立在 tabFolder 上
    final Composite comp = new Composite(tabFolder, SWT.NONE);
    tabItem_2.setControl(comp);

    final Text text_1 = new Text(comp, SWT.BORDER);
    text_1.setBounds(20, 11, 100, 25);

    final Text text_2 = new Text(comp, SWT.BORDER);
    text_2.setBounds(19, 46, 100, 25);
}
```

程序说明：

图 9.6 直观地反映了实例中各组件的嵌套关系，在阅读代码时应注意如下要点：

- TabItem 并非容器类，看 TabItem 的谱系图即知，所以 Group 是不能建立在 TabItem 中的。TabItem 和各组件一样，建立在 TabFolder 容器下。
- TabItem 的作用可以比喻为 TabFolder 中的手臂，TabFolder 通过 TabItem 来对其中的组件进行控制。
- 每一个 TabItem 只能用 setControl 方法来控制一个界面组件，所以我们才将两选项页的组件分别建立在两个容器中：两 Radio 组件建立在 Group 容器里，让 TabItem 控制 Group；两 Text 组件建立在 Composite 容器里，让 TabItem 控制 Composite。

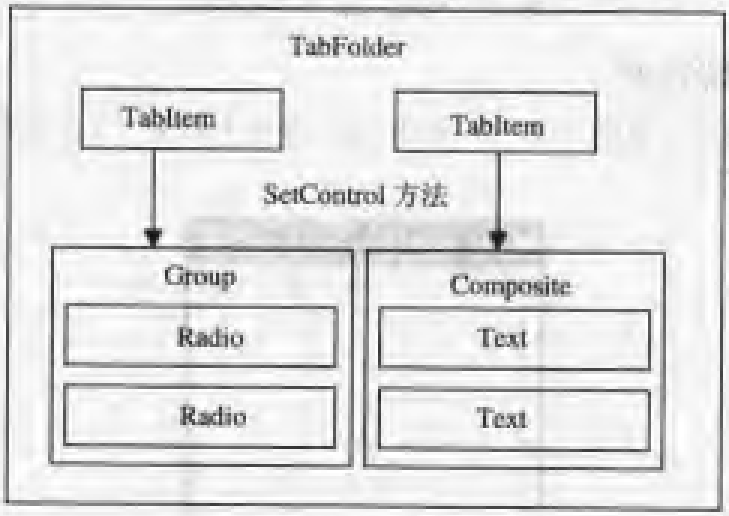


图 9.6 组件的关系示意图

9.4 分割窗（SashForm 类）

分割窗是一个很常用的组件，资源管理器（见图 9.7）的左右两个容器就是由分割窗的分割栏分开的。只需要将组件创建在分割窗容器上，它就会自动地将组件按设定好的方式分割排列好，在使用上是极其简单的，所需要的代码也很少。

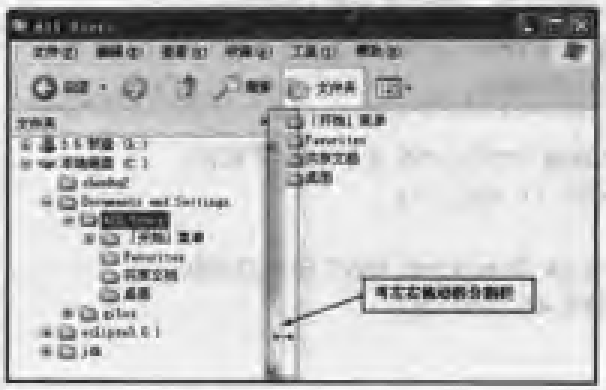


图 9.7　分割窗示意图

1．SashForm 类的谱系图

可见 SashForm 类也是 Composite 的子类，如图 9.8 所示。

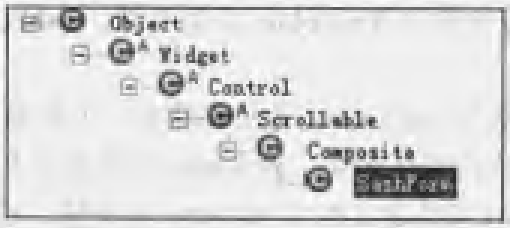


图 9.8　SashForm 类的谱系图

2．SashForm 类的实例

图 9.9 是实例的运行界面。在一个 SashForm 容器中创建三个文本框，中间的两条分割栏是可以左右拖动的。

图 9.9　SashForm 类的实例界面

实例的代码如下：

```
//--------文件名：SashForm1.java----------
SashForm sashForm = new SashForm(shell, SWT.NONE);//定义一个分割窗容器

/*在 sashForm 容器中创建三个文本框，默认是按左右顺序依次排列*/
Text text_1 = new Text(sashForm, SWT.BORDER); //在 sashForm 容器中创建一个 Text
text_1.setText("左");
//再创建两个文本框。这里将两个语句合成一个来写的简洁写法
new Text(sashForm, SWT.BORDER).setText("中");
```

```
new Text(sashForm, SWT.BORDER).setText("右");

//setWeights 方法设置了 sashForm 中各栏的宽度比例－左:中:右=3:1:2
sashForm.setWeights(new int[] { 3, 1, 2 });
sashForm.setBounds(10, 10, 206, 168);
```

9.5 带滚动条的面板（ScrolledComposite 类）

有时界面的组件会多到一个窗口都无法装下，这时 ScrolledComposite 类就派上用场了。要注意，ScrolledComposite 虽然也是 Composite 的一个子类，但不要将组件直接建立在 ScrolledComposite 中，而应将组件都建立在一个 Composite 中，然后再将 Composite 建立在 ScrolledComposite 中，也就是说用一个 Composite 来作中转。

1. ScrolledComposite 类的谱系图（如图 9.10 所示）

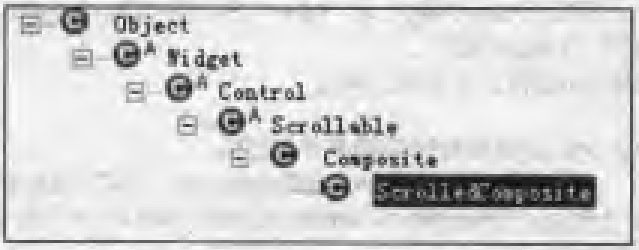


图 9.10 ScrolledComposite 类的谱系图

2. ScrolledComposite 类的实例

图 9.11 为实例的运行界面，小窗口内有八个文本框，分两列四行排放，但有一行文本框没有显示出来，这些文本框都创建在一个 Composite 中，而 Composite 又创建在 ScrolledComposite 中。由图中可以看到 ScrolledComposite 的横、竖两个滚动条，它在形状上和带滚动条的 Text 有点类似，只不过 Text 是用来容纳文字的，而 ScrolledComposite 是用来容纳界面组件的。

图 9.11 实例运行界面

为了更好地了解 ScrolledComposite、Composite 和文本框之间的关系，请看图 9.12。

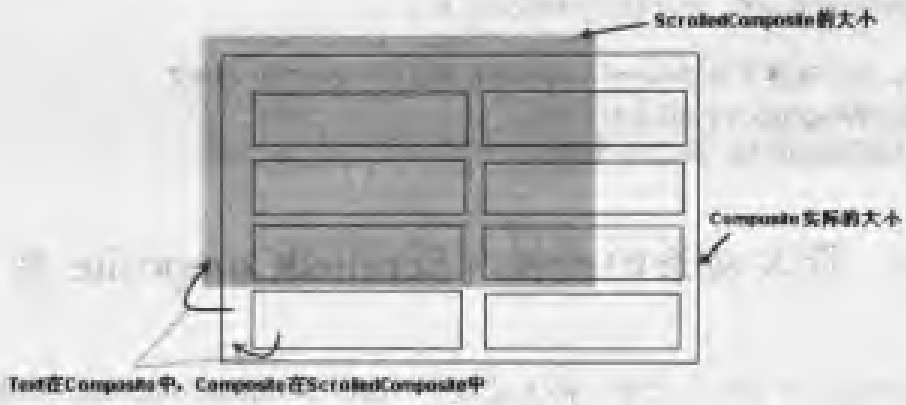


图 9.12　实例界面分析图

实例的代码如下：

```
//--------文件名：ScrolledComposite1.java---------
//定义一个 ScrolledComposite，式样为深陷型、带水平滚动条、带垂直滚动条
ScrolledComposite scrolledComposite = new ScrolledComposite(shell, SWT.BORDER |
SWT.H_SCROLL | SWT.V_SCROLL);
scrolledComposite.setBounds(12, 6, 256, 148);

//定义一个面板 Composite，用此面板来容纳其他的组件
Composite composite = new Composite(scrolledComposite, SWT.NONE);
//这里没用 setBounds 是因为 composite 和 scrolledComposite 的左上角自动重合
composite.setSize(326, 237);
//设置 composite 被 scrolledComposite 控制（指滚动条拖动）
scrolledComposite.setContent(composite);

//建立八个文本框，以横二竖四排列。这里将 Text 的定义和 Text 的定位两个语句合写在了一起
new Text(composite, SWT.BORDER).setBounds(12, 13, 132, 40);
new Text(composite, SWT.BORDER).setBounds(155, 13, 132, 40);
new Text(composite, SWT.BORDER).setBounds(10, 70, 132, 40);
new Text(composite, SWT.BORDER).setBounds(155, 70, 132, 40);
new Text(composite, SWT.BORDER).setBounds(10, 120, 132, 40);
new Text(composite, SWT.BORDER).setBounds(155, 120, 132, 40);
new Text(composite, SWT.BORDER).setBounds(10, 170, 132, 40);
new Text(composite, SWT.BORDER).setBounds(156, 171, 132, 40);
```

程序说明：

在代码中要注意 Composite 不仅要在创建时将 ScrolledComposite 作参数传入（即创建于其上），而且还要用 scrolledComposite.setContent(composite)来设置一下才会生效。

9.6　本 章 小 结

本章介绍了容器类 Composite、Group、TabFolder、SashForm、ScrolledComposite 的使用，其中 Composite 是使用最频繁的容器。一般容器类的使用都比较单一，主要是控制它们的大小，在第 10 章中容器类有更详细的使用示例。

第 10 章　布局管理器

10.1　布局管理器简介

组件装在容器里，那么这些组件的位置该如何布局呢？在这之前的所有例子都是使用 setBounds 来进行绝对坐标定位，而本章将介绍采用布局管理器的方式来布局容器中的组件。布局管理器定义了组件的摆放方式，它的概念和 AWT/SWING 的布局管理器是一样的，甚至使用方法也类似。

为什么 VB、Delphi 程序不需要布局管理器，而 Java GUI 程序却需要呢？因为 Java 开发程序的目标是要跨平台的，而每一个操作系统的屏幕定义都是不同的，如果像 VB、Delphi 那样使用绝对坐标定位界面组件，那么在 Windows 下看着好好的界面，到了 Linux 和 MacOS 下就会变得一塌糊涂。除非你开发的 Java GUI 程序只在 Windows 系统下使用，但即便如此，在不同的屏幕分辨率下，界面的形状也会大不一样。

不过有一弊必有一利，用布局管理器虽然麻烦，但开发的界面柔性较好，在不同的操作系统、不同的分辨率下，都能保持界面不变形。从本章之后就将全部用布局管理器的方式来进行界面设计编程了，因为布局管理器是 SWT 界面编程中的最重要的部分，必须要熟练掌握。

SWT 中主要的几种布局管理器如下所示：

- FillLayout，充满式布局管理器，组件的大小会尽量地充满整个容器。
- RowLayout，行列式布局管理器，简单的排列组件。
- GridLayout，网格式布局管理器，非常强大，可以胜任所有布局。
- FormLayout，表格式布局管理器，通过创建组件各个边的距离来布局组件，和 GridLayout 一样强大。
- StackLayout，堆栈式布局管理器。像书页一样，只显示最前面的组件。

图 10.1 给出了这几个布局管理器的谱系图，由图可知它们都是 Layout 类的子类。

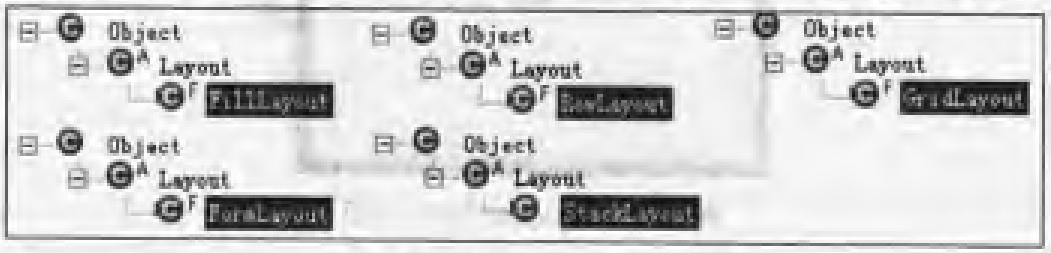


图 10.1　五个布局管理器的谱系图

10.2　充满式（FillLayout 类）

FillLayout 是最简单的布局类，它把组件摆放成一行或一列，并强制组件大小一致。一般地，组件的高度与最高组件一致，宽度与最宽组件相同。这话听起来比较绕口，简单来说，就是应用了 FillLayout 的容器，它里面的组件都会尽量地充满这个容器。另外 FillLayout 不能折行，不能设置边界距离和间距。

FillLayout 一般使用于任务栏、工具栏、Group 中的一组复选框，或者容器内只有一个组件的时候。例如，如果一个 Shell 内只有一个 Group 组件，那么将 FillLayout 用于 Shell，则其内的 Group 将完全充满 Shell。

FillLayout 虽然其功能单一，但由于其自动充满窗口的方式，在 SWT 界面设计中是很常用的。

10.2.1　FillLayout 的实例 1

一个主窗口，窗口中有一个按钮，如果要将主窗口的 Shell 容器应用 FillLayout 布局，则实现的代码如下：

```
//-------文件名：FillLayout1.java---------
shell.setLayout(new FillLayout()); //将 FillLayout 对象应用于 shell 上
new Button(shell, SWT.NONE).setText("确定");: //Button 建于 shell 中
```

在这个实例中，用 new FillLayout()定义了一个 FillLayout 对象，并将它应用在 shell 上，那么 shell 中的所有组件的大小都会尽量地撑大，挤满整个窗口，如图 10.2 所示。要注意，这里已经没有再用 setBounds 来进行绝对坐标定位。

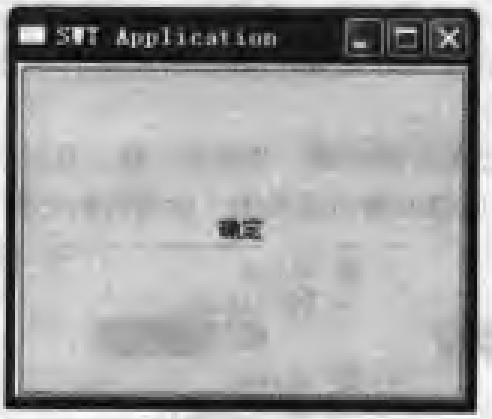


图 10.2　FillLayout 实例的界面

10.2.2　FillLayout 的实例 2

如果 shell 中有两个组件，情形将会怎样呢？例如有两个按钮，其实例代码如下：

```
//----------文件名：FillLayout2.java---------
shell.setLayout(new FillLayout());
new Button(shell, SWT.NONE).setText("确定");
new Button(shell, SWT.NONE).setText("取消");
```

运行界面如图 10.3 所示，可见两个按钮都撑大了，共同平分了 Shell 的空间。

10.2.3　使用 SWT.VERTICAL 式样更改排列方式

在 10.2.2 小节例子中的代码中，将“new FillLayout()”改为“new FillLayout (SWT.VERTICAL)”后，按钮的排列方式将由原来默认的水平排列改成了垂直排列。运行界面如图 10.4 所示。

图 10.3　Shell 中有两个按钮时的界面

图 10.4　SWT.VERTICAL 式样

10.3　行列式（RowLayout 类）

RowLayout 可以使组件折行显示，并可设置边界距离和间距。另外，它还可以对每个组件通过 setLayoutData 方法设置 RowData 对象，RowData 用来设置组件的大小。

10.3.1　RowLayout 的基本实例

```
//----文件名：RowLayout1.java---------
RowLayout rowLayout = new RowLayout();
rowLayout.marginTop = 30; //组件距容器上边缘 30 个像素的间隔
rowLayout.marginLeft = 50; //组件距容器左边缘 50 个像素的间隔
rowLayout.spacing = 3;//组件之间的间隔为 3 个像素
shell.setLayout(rowLayout); //shell 应用 RowLayout 布局
//在 shell 里建立 3 个按钮
new Button(shell, SWT.NONE).setText("b1");
new Button(shell, SWT.NONE).setText("button2");
new Button(shell, SWT.NONE).setText("button3");
```

程序说明：

marginTop 那句的注释中指明是“距容器上边缘”，这个容器是指应用了 RowLayout 布局的容器，而不是指主窗口。当然，在本例中主窗口 Shell 恰好就是这些组件的容器。

本实例的运行界面如图 10.5 所示。

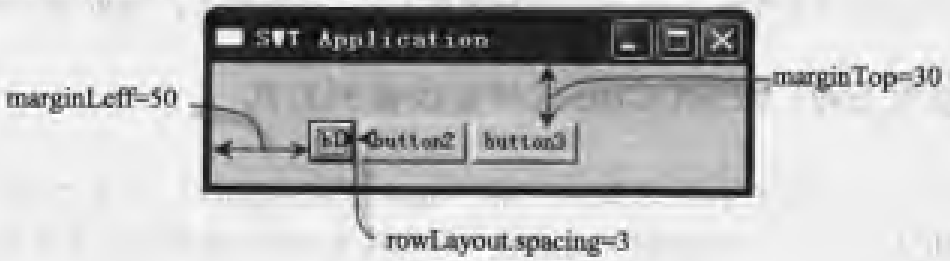


图 10.5　RowLayout 实例的运行界面

如果窗口的宽度不足时，按钮将会自动折行，如图 10.6 所示。

注意：本例中 b1 按钮和其他两个按钮的大小是不同的。如果将此例的布局管理器改用 FillLayout，则 3 个按钮将充满窗口，并且按钮大小相同，如图 10.7 所示。由这个比较也可以得知 FillLayout 和 RowLayout 的各自不同的特点了。

图 10.6　当窗口的宽度不足时的界面

图 10.7　改为 FillLayout 时的界面

10.3.2　使用 rowLayout.wrap 控制自动折行（默认值 true）

在 10.3.1 小节的例子中，当窗口的宽度不足时，按钮会自动折行。如果我们不想让它自动折行，那么该怎么设置？用以下语句即可，运行界面如图 10.8 所示。

```
RowLayout rowLayout = new RowLayout();
rowLayout.wrap=false; //false 不自动折行。默认值为 true
……//后面代码与前例（RowLayout1.java）相同，省略
```

错误的用法：

```
RowLayout rowLayout = new RowLayout(SWT.WRAP); //这样用得不到想要的界面效果
```

由错误用法的示例可以看到 SWT.*这种式样的表达方式也不是很完美，因为如果不了解某个组件可以使用哪一些式样，就有可能产生上面的误用。

图 10.8　加 rowLayout.wrap=false 语句时，窗口宽度足够和不够两种情况的界面比较

10.3.3　使用 SWT.VERTICAL 式样控制排列方式

RowLayout 默认是水平排列组件，如果在定义 RowLayout 对象时加上 SWT.VERTICAL 式样，则变为垂直排列组件，如图 10.9 所示。

用法如下：

```
new RowLayout(SWT.VERTICAL)
```

图 10.9　SWT.VERTICAL 式样及当窗口长度不断缩小时的界面效果

10.3.4　使用 rowLayout.pack 控制组件大小（默认值 true）

rowLayout.pack =false 的作用是让组件大小相同，如图 10.10 所示。

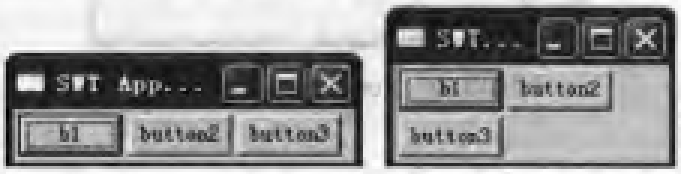


图 10.10　rowLayout. pack =false 的效果图

10.3.5　使用 rowLayout.justify 控制空间伸展（默认值 false）

rowLayout. justify = true 的作用是让组件可根据空间进行伸展，如图 10.11 所示。

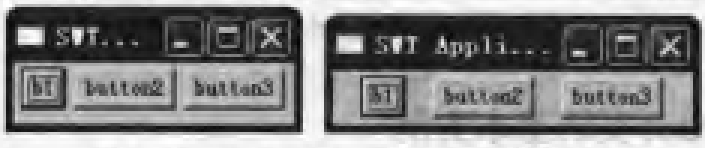


图 10.11　rowLayout. justify =true 的效果图

10.3.6　使用 RowData 更改按钮的外观

回顾上面 RowLayout 的例子，我们发现除了 rowLayout.pack =false，其他的设置只是改变按钮在容器中的布局，而对按钮的外观形状（例如大小）没有影响。

如果要改变按钮本身的外观，则需要使用 RowData 类。RowData 称为布局数据类，为 RowLayout 专用。在讲 GridLayout 类时，它也有个专用的布局数据类 GridData，GridData 的目的也是和 RowData 一样，即改变容器中组件的外观形状。

RowData 最常用的定义格式是：new RowData (int width, int height)。例如，RowData rowData=new RowData (10, 30)，表示应用了这个 rowData 的组件将改变其大小，变成宽 10 像素、高 30 像素。

我们就把 10.3.1 小节最开始的实例改动一下，代码如下：

```
//-----文件名：RowLayout2.java--------
shell.setLayout(new RowLayout());
//将 b1 按钮改为 30 像素宽，50 像素长
Button b1 = new Button(shell, SWT.NONE);
b1.setText("b1");
b1.setLayoutData(new RowData(30, 50));
new Button(shell, SWT.NONE).setText("button2");
new Button(shell, SWT.NONE).setText("button3");
```

运行界面如图 10.12 所示。

图 10.12　RowData 实例的界面

10.4　网格式（GridLayout 类）

GridLayout 可能是最常用的、功能最强大的标准布局类了，当然它也最复杂。GridLayout 的布局方式是把容器里的组件摆放在一个个格子里，它有许多可设置项，并且同 RowLayout 一样也有专用的布局数据类 GridData。GridData 的使用颇为复杂，而 GridLayout 强大之处也在于它可以通过 GridData 来设置每一个组件外观形状。

10.4.1　GridLayout 类的基本实例

这个实例中，shell 应用 GridLayout 布局方式，并在 shell 中创建 5 个按钮。通过这个实例可了解 GridLayout 的最基本的使用方式。

```
//-----文件名：GridLayout1.java---------
GridLayout gridLayout = new GridLayout();//生成一个 GridLayout 对象
shell.setLayout(gridLayout); //将 gridLayout 布局应用于 shell 面板
//建立 5 个按钮，一个按钮占一列，由左向右排，一行排完后自动转到下一行
new Button(shell, SWT.NONE).setText("b1");
new Button(shell, SWT.NONE).setText("button2");
new Button(shell, SWT.NONE).setText("b3");
new Button(shell, SWT.NONE).setText("button4");
new Button(shell, SWT.NONE).setText("button5");
```

运行界面如图 10.13 所示，用 RowLayout(SWT.VERTICAL)也可以得到这种界面效果。

图 10.13　GridLayout 类基本实例的界面

10.4.2　使用 gridLayout.numColumns 设置列数（默认值 1）

numColumns 是 GridLayout 的最重要的属性，通常它是要第一个设置的属性。例如，gridLayout.numColumns=2，是指把容器分为两列，组件从左到右摆放在列里，这时第三个组件将自动添加新的一行。

具体的代码如下：

```
GridLayout gridLayout = new GridLayout();
gridLayout.numColumns = 3; //相应改为 1、2、3
shell.setLayout(gridLayout);
```

还有一种写法，就是用 GridLayout 的另一个构造函数，代码如下：

```
shell.setLayout(new GridLayout(3,false));
```

图 10.14 给出了当 numColumns 设为 1、2 或 3 时的效果。

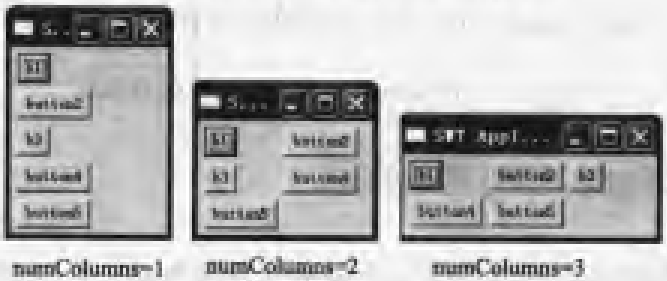


图 10.14　不同的 numColumns 值的效果图

10.4.3　使用 makeColumnsEqualWidth 把组件等距分开（默认值 false）

在 10.4.2 小节的例子中，组件都是向左紧挨着排列，如果加一句“gridLayout.makeColumnsEqualWidth = true;”，则组件会按等距分开，它实际上是因为分割的列宽度增加了，这时组件在每一列中还是向左对齐（右对齐、居中的方法见下面关于 GridData 的内容），效果如图 10.15 所示，右边两图是窗口变化大小时的效果。

图 10.15　makeColumnsEqualWidth = true 的效果图

10.4.4　使用 MarginWidth、MarginHeight、HorizontalSpacing、VerticalSpacing 属性

GridLayout 边距和间距的设置与 RowLayout 的类似：

- 同样可以分别设置 horizontalSpacing 和 verticalSpacing（RowLayout 中的间距根据组件是水平排列或垂直排列来设置水平间距或垂直间距）。
- 不同的是，左边距和右边距统一成 marginWidth，上边距和下边距统一成 marginHeight。

gridLayout.horizontalSpacing = 100;（默认值 5）是设置了列之间的间隔为 100 像素，如图 10.16 所示。verticalSpacing 属性与 horizontalSpacing 类似，只不过 verticalSpacing 是设置垂直间隔距离。

图 10.16　horizontalSpacing = 100 的效果图

10.4.5　使用 GridData 控制复杂布局

仅前面所讲的内容还是不能胜任现实中复杂的界面布局，它们都只能控制容器的布局，例如，将容器分成几列，每列的间隔多少，列是否平分总宽度等。但容器中的组件的外观

形状并没有改变，例如，让一个按钮加宽到占用两个列，或加长到占用两行，让按钮跳过一列排到下一行等。有些读者可能会想用按钮本身的设置长宽的方法来实现这些效果，但这样虽然可以让按钮变宽变长，但同时也会影响其他组件的布局，并不能真正达到我们要的总体效果。

GridLayout 与 GridData 相结合可以变化出很多复杂的布局方式。需要注意的是，GridData 只用于 GridLayout 布局下的的组件，像 FillLayout，RowLayout 布局下的组件不能用这个类，也就是说，GridData 是 GridLayout 专用的，就像 RowData 是 RowLayout 专用的一样。另外还要注意：一个 GridData 对象只能用于一个组件，GridData 是不能多个组件共用的，否则将会引起界面的紊乱。

以下将针对一些常用的布局来演示 GridData 的用法。

1．horizontalSpan

horizontalSpan 是设置组件占用的列数。给出一个实例如下：

```
//----------文件名：GridData1.java----------
shell.setLayout(new GridLayout(2,false));//2 列
new Button(shell, SWT.NONE).setText("b1");
new Button(shell, SWT.NONE).setText("button2");
Button button = new Button(shell, SWT.NONE);
GridData gridData = new GridData();//定义一个 GridData 对象
gridData.horizontalSpan = 2;//让 button 抢占两个列的空间
button.setLayoutData(gridData);
button.setText("b3");
new Button(shell, SWT.NONE).setText("button4");
new Button(shell, SWT.NONE).setText("button5");
```

运行界面如图 10.17 所示。

图 10.17　horizontalSpan = 2 的效果图

2．GridData.HORIZONTAL_ALIGN_FILL（对齐式充满）

在图 10.17 中可以很明显看到 b3 按钮占据了两列布局空间（方框标出的部分）。这里有些读者可能会困惑：既然 b3 占了两列空间，应该变得长一些才对，为什么它还是老样子呢？我们可以这样来理解，组件拥有多少空间和它真正在显示上占了多少空间是两回事。打个比喻，有个地主有两亩地，但他就种一亩，另一亩自己不种也不给别人种，他就让这多余的一亩荒着（这地主属于偏执狂）。

如果想让 b3 在显示上也占两列，即让 b3 充满在自己的空间中，怎么办？非常简单，将定义 GridData 的一句改为“new GridData(GridData.HORIZONTAL_ALIGN_FILL);”即可。

GridData.HORIZONTAL_ALIGN_FILL 是一个 int 常量，叫“水平对齐式充满”，效果如图 10.18 所示。

图 10.18　GridData.HORIZONTAL_ALIGN_FILL 式样的效果图

3．GridData.FILL_HORIZONTAL（水平抢占式充满）

它会像饿狼一样将所有水平方向的空闲空间都收为己有，图 10.19 为将此式样应用到 b3 按钮上的效果。

图 10.19　GridData.FILL_HORIZONTAL 的效果图

有人会想，b3 充满整个窗口了，button5 也应该齐头并进才好，这叫“比翼双飞”。很简单，只要将设置 b3 的方法同样用到 button5 上即可，即把原来的“new Button(shell, SWT.NONE).setText("button5");”改成如下语句：

```
Button button5 = new Button(shell, SWT.NONE);
GridData gridData2 = new GridData(GridData.HORIZONTAL_ALIGN_FILL);
button5.setLayoutData(gridData2);
button5.setText("button5");
```

运行效果如图 10.20 所示。

图 10.20　button5 跟进的效果图

4．GridData.FILL_BOTH（双向抢占式）

它相当于 FILL_HORIZONTAL（水平抢占式）、FILL_VERTICAL（垂直抢占式）合并的效果，如图 10.21 所示。

5．gridData.horizontalAlignment、gridData.verticalAlignment

图 10.21　GridData.FILL_BOTH 的效果图

这两个属性用来设置组件在格子里的对齐方式。使用方法如下：

```
gridData.horizontalAlignment = GridData.CENTER;
```

图 10.22 给出了 horizontalAlignment 设置不同值的效果，verticalAlignment 与此类似，故省略。

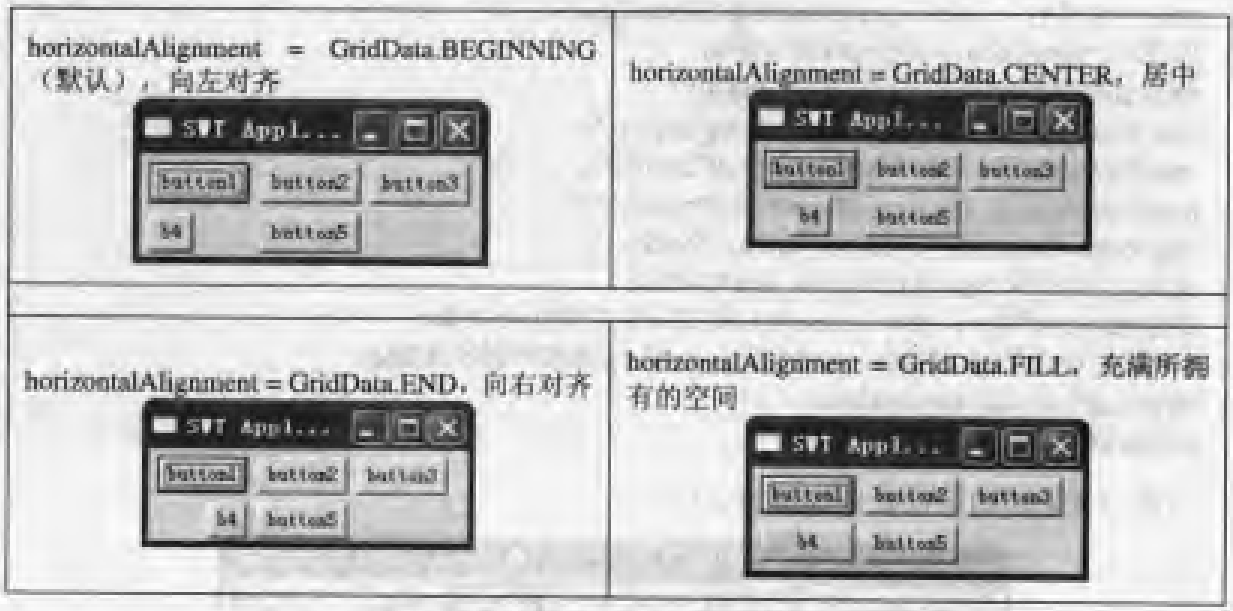


图 10.22　HorizontalAlignment 的效果图

6．gridData.horizontalIndent

horizontalIndent 属性可以使组件向右移动指定长度，此属性只有当 horizontalAlignment 设为 BEGINNING（默认值）时有效。以下代码把 b5 右移动 10 个像素，如图 10.23 所示。

```
GridData gridData = new GridData();
gridData.horizontalIndent = 10;
b5.setLayoutData(gridData);
```

图 10.23　gridData.horizontalIndent 效果图

7．gridData.grabExcessHorizontalSpace、gridData.grabExcessVerticalSpace（默认值flase）

这两个属性主要用在 Text、List 中，效果是：当组件所在的容器大小改变时，组件的占有空间也自动增大或减小。例如，设置一个 Text 组件为 grabExcessHorizontalSpace=true，那么当用户调整窗口，导致容器的宽度变化时，Text 所在列空间也会随着变化，但同一行上的其他组件会保持宽度不变。

这里要强调一点：改变的是它们占有的空间，但它们本身的大小不会改变，要让它们本身的大小也随着改变，还要加一句“gridData.horizontalAlignment = GridData.FILL”。这个效果和用前面讲的 GridData.FILL_HORIZONTAL、GridData.FILL_VERTICAL 式样差不多，一般都还是用 GridData.FILL_HORIZONTAL，这样可以节省两个语句。

给出一个 b5 应用 grabExcessHorizontalSpace 属性的实例，代码如下：

```
//-------文件名：GrabExcessHorizontalSpace1.java---------
//3 列且 makeColumnsEqualWidth=false
shell.setLayout(new GridLayout(3, false));
new Button(shell, SWT.NONE).setText("button1");
new Button(shell, SWT.NONE).setText("button2");
new Button(shell, SWT.NONE).setText("button3");
new Button(shell, SWT.NONE).setText("button4");
Button button = new Button(shell, SWT.NONE);
GridData gridData = new GridData();//定义一个 GridData 对象
gridData.grabExcessHorizontalSpace = true; //空间横向随容器变化
button.setLayoutData(gridData);
button.setText("b5");
```

当窗口变化时的运行效果如图 10.24 所示。

图 10.24　GrabExcessHorizontalSpace=true 时效果图

如果将例子中的 gridData.horizontalAlignment = GridData.FILL 的注释去掉，则运行效果如图 10.25 所示，这个效果与 GridData.FILL_HORIZONTAL 式样一样。

```
Button button = new Button(shell, SWT.NONE);
GridData gridData = new GridData();//定义一个 GridData 对象
gridData.grabExcessHorizontalSpace = true; //空间横向随容器变化
gridData.horizontalAlignment = GridData.FILL; //充满所拥有的空间
button.setLayoutData(gridData);
button.setText("b5");
```

8．gridData.widthHint、gridData.heightHint

gridData.widthHint 和 gridData.heightHint 属性可以改变组件的宽度和高度，但前提是不与 GridLayout 的其他设置相矛盾。以下例子代码为改变 b5 为 200 像素长和 15 像素高，运

行效果如图 10.26 所示。

图 10.25　加上 gridData.horizontalAlignment = GridData.FILL 的效果图

```
Button button = new Button(shell, SWT.NONE);
GridData gridData = new GridData();
gridData.widthHint = 100;
gridData.heightHint = 15;
button.setLayoutData(gridData);

button.setText("b5");
```

图 10.26　gridData.WidthHint、gridData.HeightHint 的效果图

在图中的第三个窗口可以看到，无论窗口变多大，按钮的长度始终是 100 像素。但如果 gridData.widthHint、gridData.heightHint 的设置与 GridLayout 的其他设置相矛盾会怎么样呢？下面的代码应用了 FILL_HORIZONTAL 式样，效果如图 10.27 所示：第一个窗口按钮保持 100 像素，不会随窗口缩小，因而被遮住了；第三个窗口，当窗口变大时按钮长度也跟着变大，不再是固定 100 像素长了。

```
Button button = new Button(shell, SWT.NONE);
GridData gridData = new GridData(GridData.FILL_HORIZONTAL);
gridData.widthHint = 100;
gridData.heightHint = 15;
button.setLayoutData(gridData);
button.setText("b5");
```

图 10.27　加上 GridData.FILL_HORIZONTAL 式样后的效果图

10.5　堆栈式（StackLayout 类）

StackLayout 称做堆栈式布局。所谓堆栈式，就是像一副叠在一起的扑克牌，一叠 UI 面板只显示位于最上面的那个面板。图 10.28 是 Eclipse 中比较典型的用到堆栈式布局的界

面，右边部分就是一个用了堆栈式布局的容器，当左边选不同的项时，右边部分就显示这一项相应的设置组件。

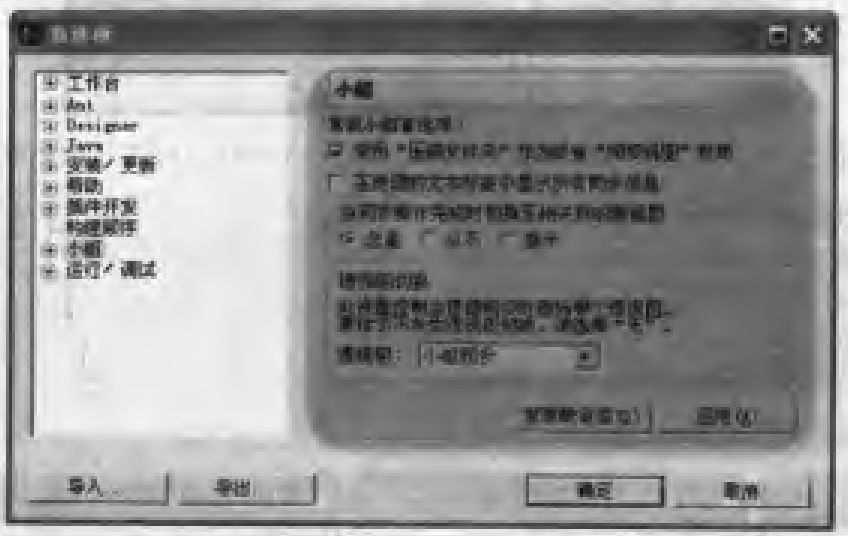


图 10.28　一个堆栈式布局的实际界面

StackLayout 布局的使用是比较简单的，下面我们就来写一个用 StackLayout 的例子。其效果如图 10.29 所示，而图 10.30 是它的界面分析图。

图 10.29　StackLayout 实例的初始界面（左）、单击 Text1（中）和 Text2（右）按钮后的界面

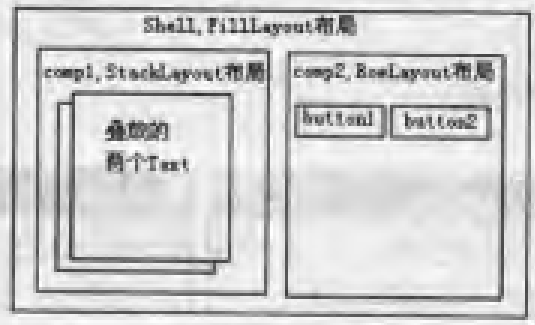


图 10.30　界面分析图

实例代码如下：

```
//----------文件名：StackLayout1.java----------
shell.setLayout(new FillLayout()); //shell 应用充满式布局
//因为 comp1 容器使用 stackLayout 布局，所以 comp1 中的两个 Text 框是重叠在一起的
final Composite comp1 = new Composite(shell, SWT.NONE);
final StackLayout stackLayout = new StackLayout();
```

```
comp1.setLayout(stackLayout);
//在 comp1 中创建两个 Text 框
final Text text1 = new Text(comp1, SWT.BORDER);
text1.setText("Text1");
final Text text2 = new Text(comp1, SWT.BORDER);
text2.setText("Text2");

//在一个新容器中建立两个按钮用来控制上面两个 text 框的显示
Composite comp2 = new Composite(shell, SWT.NONE);
comp2.setLayout(new RowLayout());
//按钮 button1，它控制 Text1 的显示
Button button1 = new Button(comp2, SWT.NONE);
button1.setText("显示 Text1");
button1.addSelectionListener(new SelectionAdapter() { //单击事件
public void widgetSelected(SelectionEvent e) {
        /**********************************************
         * --------下面两句是 StackLayout 中控制的关键----------
         **********************************************/
        stackLayout.topControl = text1; //将 t1 框显示
        comp1.layout(); //将界面刷新一下,否则显示上会没有任何反映的
    }
});
//按钮 button2,它控制 Text2 的显示
Button button2 = new Button(comp2, SWT.NONE);
button2.setText("显示 Text2");
button2.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        stackLayout.topControl = text2;
        comp1.layout();
    }
});
```

10.6　表格式（FormLayout 类）

FormLayout 也是一种和 GridLayout 一样强大，而且还很灵活、精确的布局，这个布局是 SWT 2.0 版新增的。通常 GridLayout 与 FormLayout 可以做成同样的效果，但有时使用后者会更有效，不会像 GridLayout 因容器大小变化而导致布局错位。以下将通过实例来介绍 FormLayout 布局的使用。

10.6.1　使用 marginWidth、marginHeight 设置边距

这两个属性用来设置容器的左边距和上边距（单位：像素）。下面给出了一个示例，

其效果如图 10.31 所示。

```
//--------文件名：FormLayout1.java----------
FormLayout formLayout = new FormLayout();
formLayout.marginWidth = 100;
formLayout.marginHeight = 50;
shell.setLayout(formLayout);
new Button(shell, SWT.NONE).setText("button1");
```

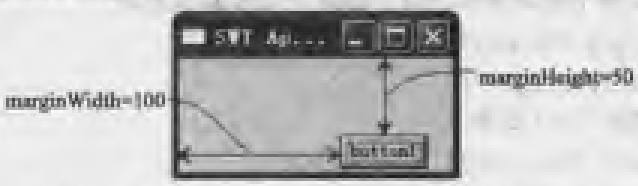


图 10.31　FormLayout 类的基本实例的效果图

10.6.2　使用 FormData 的构造函数

FormLayout 也有自己的布局数据类 FormData，它的使用方法是：new FormData ()或 new FormData (int width, int height)，如下示例，效果如图 10.32 所示。

```
//--------文件：FormData1.java----------
shell.setLayout(new FormLayout());
//new FormData ()
Button button1 = new Button(shell, SWT.NONE);
button1.setText("button1");
FormData formData = new FormData();
button1.setLayoutData(formData);
//new FormData (int width, int height)，单位：像素
Button button2 = new Button(shell, SWT.NONE);
button2.setText("button2");
FormData formData2 = new FormData(200, 50);//button2 变成 200 长，50 宽
button2.setLayoutData(formData2);
```

图 10.32　FormData 不同构造函数之间的效果图

10.6.3　FormAttachment 类的用法

FormAttachment 是在 FormData 下的，更进一步地布局数据类，它的用法主要体现在它不同的构造函数中。

1．new FormAttachment (int numerator, int offset)

如下例，button1 的顶边（formData.top）离 shell 容器的空白边距是 shell 容器总体空白

长度的 60%，偏移的点数（points）为 0。效果如图 10.33 所示。

```
//-------文件名：FormData_FormAttachment1.java--------
shell.setLayout(new FormLayout());
new Text(shell, SWT.BORDER).setText("text1");
//将 button1 应用 FormData
Button button1 = new Button(shell, SWT.NONE);
button1.setText("button1");
FormData formData = new FormData();
formData.top = new FormAttachment(60, 0); // button1 的顶端应用 FormAttachment 设置
button1.setLayoutData(formData);
```

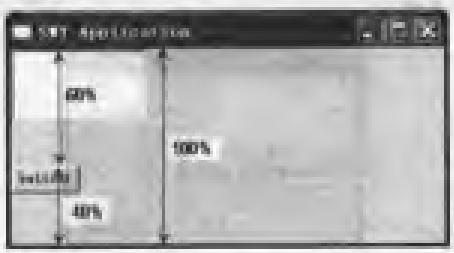


图 10.33　FormAttachment 效果图

如果将 new FormAttachment(60, 0)改为 new FormAttachment(60, 10)则效果如图 10.33 所示。从图中可见，FormAttachment(60, 10)是先按 FormAttachment(60, 0)的方式布局后，再下移 10 个像素，这里有一个布局的次序。

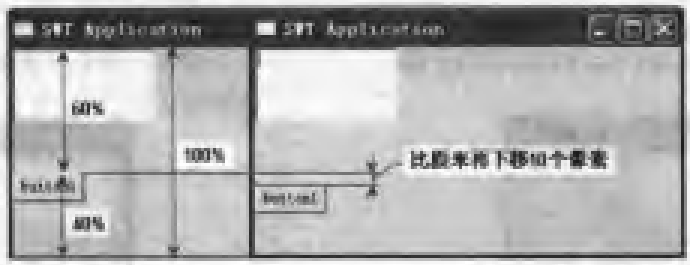


图 10.34　new FormAttachment(60, 10)的效果说明图

new FormAttachment (int numerator)相当于 new FormAttachment (int numerator, int offset) 当 offset=0 时。

new FormAttachment (int numerator, int offset)相当于 FormAttachment (int numerator, int denominator, int offset) 当 denominator = 100 时，其中 denominator 是分母，例如 FormAttachment (30, 50, 0)就是长度比例为 30/50=60%，也就是和 FormAttachment (60, 0) 的效果一样。

2．new FormAttachment (Control control, int offset, int alignment)

参数 1 是一个 Control 类，一般在使用的时候都传入一个组件（如文本框）来做参数。应用此 FormAttachment 的组件将依据参数 1 的 control 为基准来布局，offset 为离 control 的偏移量（单位：像素），alignment 为对齐方式。下面给出一个示例，效果如图 10.35 所示。

```
//-------文件名：FormData_FormAttachment2.java-------
```

```
shell.setLayout(new FormLayout());
Text text1 = new Text(shell, SWT.BORDER);
text1.setLayoutData(new FormData(100, 50));
//定义并设置 FormData
FormData formData = new FormData();
//以 text1 为基准偏移 50 像素
FormAttachment formAttachment = new FormAttachment(text1, 50);
formData.top = formAttachment;
formData.left = formAttachment;
//将 button1 应用 FormData
Button button1 = new Button(shell, SWT.NONE);
button1.setText("button1");
button1.setLayoutData(formData);
```

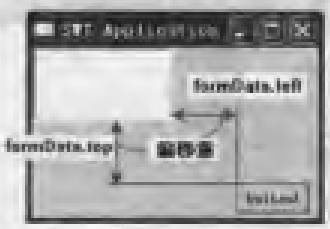


图 10.35　FormAttachment (Control control, int offset)的效果图

new FormAttachment(text1, 50，int alignment)中 alignment 的设置的效果如图 10.36 所示，表中效果的程序是以上例“FormData_FormAttachment2.java”为基础修改“FormAttachment formAttachment = new FormAttachment(text1, 50);”这一句而得。

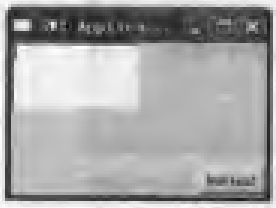
new FormAttachment(text1, 50)，不带alignment参数相当于FormAttachment (text1, 50, SWT.DEFAULT); 	alignment= SWT.CENTER（button1与text1的中心重合，因此在效果图中看不到button1，它被遮住了）
alignment= SWT.LEFT，以text1的左边界为基准偏移	alignment= SWT.RIGHT，以text1的右边界为基准偏移
alignment= SWT.TOP，以text1的上边界为基准偏移	alignment= SWT.BOTTOM，以text1的下边界为基准偏移

图 10.36　alignment 的效果图

10.7　布局的综合实例

前面所讲都还是单一的知识点，所谓“不积细流无以成江河”，界面的开发就是由这一个个小知识点的运用堆积而成。在界面开发中要充分借助 SWT Designer 工具来帮助我们快速生成初步的代码，然后再将这些代码移植到实际开发程序中。

对于一个复杂的界面，应使用“由粗到精”的设计方法，先借助容器类搭好界面框架，然后再往容器类中放置组件，最后才写事件处理等代码。切忌在一个容器中放入所有组件，如果单个容器的组件布局太过复杂，以后修改维护起来就会很困难。

下将模仿 QQ 的个人设置界面来作为本节的实例，图 10.37 是实例的运行效果。

图 10.37　实例运行效果图

首先搭好框架，为了在设计时看起来方便，可以先将 Composite 设置成 BORDER（深陷型），让 Composite 显示出来，这样就可以看到 Composite 的大小和位置，等设计好后再将它们改为 SWT.NONE 式样。

总体界面的设计思路如图 10.38 所示。

图 10.38　总体界面分析图

在程序里实现了上面框架界面后，再细化更进一步的界面，最后再将“个人资料”的界面设计出来。在界面设计中考虑和分析界面的构成方式是很重要的，一个稍微复杂一点的界面会有很多种设计方案，不同的方案虽然都可以得出最后的界面效果，但好方案和差方案在易编程性、易维护性等方面的差别却是很大的。

个人资料的界面方方正正，正适合使用 GridLayout 来布局，用 GridLayout 首先要统计一下界面的列数，这个列数要照顾到所有行的使用。个人资料里“性别”那一行的所须列数最大（6 列），因此将个人资料的容器面板设成 6 列。由于主要是讲界面的布局，所以一些下拉框的数据填充和一些数据处理程序就省略不写了。

具体的代码如下：

```
//----------文件名：QQ.java----------
public class QQ {
    /**
    * 为了自定义监听器类 MyMouseLister 中的代码能访问到以下对象,故将这些对象定义成
    * 类的实例变量
    */
    private StackLayout stackLayout = new StackLayout();
    private Composite yourDataComp;
    private Composite otherComp;
    private List selectList;
    private Composite rightComp;
    /**
    * 由于自定义方法较多程序较长，这次的主程序框架选择了 Eclipse
    * “新建”向导的第二种：“public open() mothd”
    */
    public static void main(String[] args) {
        try {
            QQ window = new QQ();
            window.open();
        } catch (Exception e) { e.printStackTrace(); }
    }

    public void open() {
        Display display = Display.getDefault();
        Shell shell = new Shell();
        shell.setSize(550, 350);
        shell.setText("个人设置");
        shell.setLayout(new GridLayout());
        {
            //分割窗口
            SashForm sashForm = new SashForm(shell, SWT.BORDER);
            sashForm.setLayoutData(new GridData(GridData.FILL_BOTH));
            {
                //分割窗左边的列表框
```

```
            selectList = new List(sashForm, SWT.BORDER);
            //作为演示只加了两项
            selectList.setItems(new String[] { "个人资料", "联系方式" });
            //加一个鼠标监听器
            selectList.addMouseListener(new MyMouseLister());//
        }
        {
            //右边的堆栈式面板
            rightComp = new Composite(sashForm, SWT.NONE);
            rightComp.setLayout(stackLayout);
            //共两页，将生成此面板的代码提出成一个方法，保证代码结构的清晰
            yourDataComp = createYourDataComp(rightComp);//个人资料的面板
            otherComp = createOtherComp(rightComp);//联系方式的面板
            //在堆栈面板上先显示“个人资料”界面
            stackLayout.topControl = yourDataComp;
        }
        //分割窗口的左右空间比例
        sashForm.setWeights(new int[] { 1, 4 });
    }
    {
        //界面的按钮的面板
        Composite buttonComp = new Composite(shell, SWT.BORDER);
        //使用 GridData 设置 buttonComp 在它父容器 Shell 中的布局方式
        GridData gridData = new GridData();
        gridData.horizontalAlignment = GridData.END;//让 buttonComp 向右靠
        buttonComp.setLayoutData(gridData);
        //设置 buttonComp 的布局为 RowLayout，用来设定 buttonComp 内组件的布局方式
        RowLayout rowLayout = new RowLayout();
        rowLayout.spacing = 15;//按钮之间间隔 15 个像素
        buttonComp.setLayout(rowLayout);
        //在 buttonComp 下建立三个按钮
        new Button(buttonComp, SWT.NONE).setText("    确定    ");
        new Button(buttonComp, SWT.NONE).setText("    取消    ");
        new Button(buttonComp, SWT.NONE).setText("    应用    ");
    }
    shell.layout();
    shell.open();
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
}

/** 个人资料面板的生成 */
private Composite createYourDataComp(Composite rightComp) {
    Composite composite = new Composite(rightComp, SWT.NONE);
```

```
        composite.setLayout(new GridLayout(6, false));//个人资料面板分成 6 列

        //————用户号码的标签及其文本框
        new Label(composite, SWT.NONE).setText("用户号码：");
        //只读型的文本框
        Text text = new Text(composite, SWT.READ_ONLY | SWT.BORDER);
        //水平抢占式充满,并占用三列的空间. createGridData 是自定义方法
        text.setLayoutData(createGridData(GridData.FILL_HORIZONTAL, 3));

        //————图片部分，我们再用一个画板嵌套来管理
        Composite photoComp = new Composite(composite, SWT.BORDER);
        //水平垂直对齐式充满,横占两列,竖占 4 行
        photoComp.setLayoutData(createGridData(GridData.HORIZONTAL
                                _ALIGN_FILL | GridData.VERTICAL_ALIGN_FILL, 2, 4));
        photoComp.setLayout(new GridLayout(2, false));//分 2 列
        {
            //图片. Image 类的使用暂时还没有讲到,我们先用一个 Composite 来代替图片
            Composite tempComp = new Composite(photoComp, SWT.BORDER);
            tempComp.setLayoutData(new GridData(50, 50));//设定大小:宽 50,高 50
            //选择图片的箭头型按钮,并设置它向下靠
            Button selPhotoButton=new Button(photoComp,SWT.ARROWISWT.DOWN);
            selPhotoButton.setLayoutData(new GridData(GridData.VERTICAL
                                                    _ALIGN_END));
            //升级成为会员的按钮，横占 photoComp 的两列,并横向对齐充满
            Button updateButton = new Button(photoComp, SWT.NONE);
            updateButton.setLayoutData(createGridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL, 2));
            updateButton.setText("升级成为会员");
        }

        new Label(composite, SWT.NONE).setText("用户昵称：");
        Text nicknameText = new Text(composite, SWT.BORDER);
        nicknameText.setLayoutData(createGridData(GridData.HORIZONTAL
                                                _ALIGN_FILL, 3));
        new Label(composite, SWT.NONE).setText("个性签名：");
        Text attachNameText = new Text(composite, SWT.BORDER);
        attachNameText.setLayoutData(createGridData(GridData.HORIZONTAL
                                                _ALIGN_FILL, 3));
        new Label(composite, SWT.NONE).setText("等　　级：");
        {
            //Image 类的使用暂时还没有讲到,我们先用一个 Composite 来代替图片
            Composite tempComp = new Composite(composite, SWT.BORDER);
            GridData gridData = new GridData(GridData.FILL_HORIZONTAL | GridData.VERTICAL_
ALIGN_BEGINNING);
            gridData.horizontalSpan = 3;
                //Composite 默认的高度太高,故手工设定高度为 20 像素
```

```
            gridData.heightHint = 20;
            tempComp.setLayoutData(gridData);
        }

        new Label(composite, SWT.NONE).setText("性　　别：");
        Combo sexCombo = new Combo(composite, SWT.NONE);

        new Label(composite, SWT.NONE).setText("姓名：");
        Text nameText = new Text(composite, SWT.BORDER);
        nameText.setLayoutData(new GridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL));
        new Label(composite, SWT.NONE).setText("年龄：");
        Text oldText = new Text(composite, SWT.BORDER);
        oldText.setLayoutData(new GridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL));
        new Label(composite, SWT.NONE).setText("毕业院校：");
        Text schoolText = new Text(composite, SWT.BORDER);
        schoolText.setLayoutData(createGridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL, 3));
        new Label(composite, SWT.NONE).setText("生肖：");
        Combo animalCombo = new Combo(composite, SWT.NONE);
        animalCombo.setLayoutData(new GridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL));
        new Label(composite, SWT.NONE).setText("职业：");
        Text jobText = new Text(composite, SWT.BORDER);
        jobText.setLayoutData(createGridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL, 3));
        new Label(composite, SWT.NONE).setText("星座：");
        Combo constellationCombo = new Combo(composite, SWT.NONE);
        constellationCombo.setLayoutData(new GridData(GridData.HORIZONTAL
                                                    _ALIGN_FILL));
        Label introLabel = new Label(composite, SWT.NONE);
        //默认是居中，改为顶端对齐
        introLabel.setLayoutData(new GridData(GridData.VERTICAL
                                                    _ALIGN_BEGINNING));
    introLabel.setText("个人说明：");
    Text introText=new Text(composite,SWT.BORDER|SWT.WRAP);//WRAP 自动换行
        introText.setLayoutData(createGridData(GridData.HORIZONTAL
                                _ALIGN_FILL | GridData.FILL_VERTICAL, 5));
        //返回个人资料面板 composite
        return composite;
    }

    /** 生成一个简单的联系方式的面板 */
    private Composite createOtherComp(Composite rightComp) {
        Composite composite = new Composite(rightComp, SWT.NONE);
```

```
        composite.setLayout(new FillLayout());
        new Label(composite, SWT.NONE).setText("联系方式面板");
        return composite;
    }

    /** 生成 GridData 对象的重复代码太多，写成一个方法可以减少程序的行数，用起来也方便
     * 而且还要注意一个 GridData 只能被一个组件使用，不能两个组件使用一个 GriData，则
     * 有一个组件将不会显示出来 */
    private GridData createGridData(int style, int horizontalSpan) {
        GridData gridData = new GridData(style);
        gridData.horizontalSpan = horizontalSpan;
        return gridData;
    }

    private GridData createGridData(int style, int horizontalSpan
    , int verticalSpan) {
        GridData gridData = new GridData(style);
        gridData.horizontalSpan = horizontalSpan;
        gridData.verticalSpan = verticalSpan;
        return gridData;
    }

    /** 鼠标监听器,采用事件的命名内部类的写法 */
    private class MyMouseLister extends MouseAdapter {
        public void mouseDown(MouseEvent e) {
            //得到列表的当前选择项的序号(即单击的那项)
            int selectionIndex = selectList.getSelectionIndex();
            //如果单击第一项“个人资料”,则让相对应的面板移至最上面
            if (selectionIndex == 0)
                stackLayout.topControl = yourDataComp;
            else
                stackLayout.topControl = otherComp;
            rightComp.layout();//刷新堆栈式布局的顶容器
        }
    };
}
```

10.8 本章小结

FillLayout、RowLayout 功能单一，但使用也简单方便。FormLayout 组件精确，擅长于要求精细的界面，但它写起来较繁琐。GridLayout 较 FormLayout 用来方便些且一样强大，适用于方方正正的界面，它以打格子的方式来布局。虽然 GridLayout 或 FormLayout 中任何一个都有能力完成所有页面布局，但还是应该充分了解各布局方式的特性，选用对于所开

发的界面最适合的一种。

就笔者的开发实践来说，用得最多的是 GridLayout，其次是 FillLayout，RowLayout 很少用，FormLayout 基本没用过。当然不是指 FormLayout 不好，而是根据自己的使用习惯和对布局的熟悉程度来选择的。

第 11 章　其他 SWT 组件

11.1　工具栏（ToolBar 类、ToolItem 类、ViewForm 类）

本节将把创建一个简单的编辑器作为示例，如图 11.1 所示。这个编辑器有一个工具栏，工具栏上有两个按钮，单击“取得”按钮将弹出一个包含所填文字的提示窗，单击“清除”按钮可将文本框中文字清除。

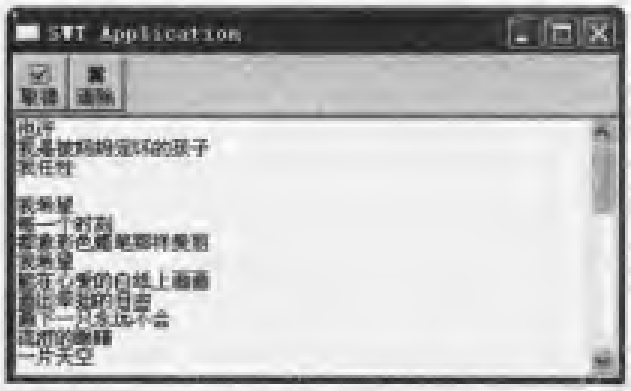


图 11.1　界面效果图

建立这个编辑器将用到三个新类：ViewForm、ToolBar 和 ToolItem，它们的谱系如图 11.2 所示。

图 11.2　谱系图

ViewForm 是一个容器，它是编辑器的基座，用来容纳工具栏和文本框。ToolBar 是一个工具栏，同时也是一个容器，即容纳工具栏的一个个按钮（ToolItem）。如图 11.3 所示。

图 11.3　示意图

代码如下：

```
//--------文件名：SimpleEditor1.java-------
shell.setLayout(new FillLayout());
//用 ViewForm 做底层容器，并用充满型布局
ViewForm viewForm = new ViewForm(shell, SWT.NONE);
viewForm.setLayout(new FillLayout());
//在 ViewForm 上建立工具栏和文本框
ToolBar toolBar = new ToolBar(viewForm, SWT.NONE);
final Text text = new Text(viewForm, SWT.BORDER | SWT.V_SCROLL);
//设置工具栏和文本框在 ViewForm 上的位置
viewForm.setContent(text);
viewForm.setTopLeft(toolBar);
/*
 * 为工具栏添加按钮及相应事件
 */
ToolItem toolItem = new ToolItem(toolBar, SWT.PUSH);
toolItem.setText("取得");
//按钮图片
toolItem.setImage(new Image(display, "icons/selectAll.gif"));
toolItem.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
            String str = text.getText(); //取得文本框的文字
            MessageDialog.openInformation(null, null, str);
    }
});

ToolItem toolItem2 = new ToolItem(toolBar, SWT.PUSH);
toolItem2.setText("清除");
toolItem2.setImage(new Image(display, "icons/remove.gif"));
toolItem2.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
            text.setText("");//清除文本框中的文字
    }
});
```

现将 ToolItem 的式样罗列如下：

- SWT.PUSH，普通按钮，本例即用此式样。
- SWT.CHECK，单击按钮则下陷，再单击才会弹起来。
- SWT.RADIO，一组 RADIO 类型的按钮，如果一个按下，其他的都会弹起来。
- SWT.SEPARATOR，按钮间的分隔线。
- SWT.DROP_DOWN，旁边带个下拉条的按钮。

11.2　动态工具栏（CoolBar 类、CoolItem 类）

CoolBar 组件是一个可以放置多个工具栏的容器，基于 CoolBar 做出的工具栏可以通过拖动操作改变排放位置，如图 11.4 所示。我们把这种工具栏称为 CoolBar 型工具栏。

图 11.4　CoolBar 型工具栏

CoolBar 是一个容器类，它需要结合其子项 CoolItem 来使用，其谱系如图 11.5 所示。

图 11.5　谱系图

由于 CoolBar、CoolItem 与 11.1 节讲的 ToolBar、ToolItem 名称是如此对应相似，所以很容易产生这样的错误理解：在代码中用 CoolBar 替换 ToolBar，用 CoolItem 替换 ToolItem 就可以得到图 11.4 所示的 CoolBar 型工具栏。实际上，要得到 CoolBar 型工具栏必须是 CoolBar、CoolItem、ToolBar 和 ToolItem 四者一起使用，它们的关系如图 11.6 所示。

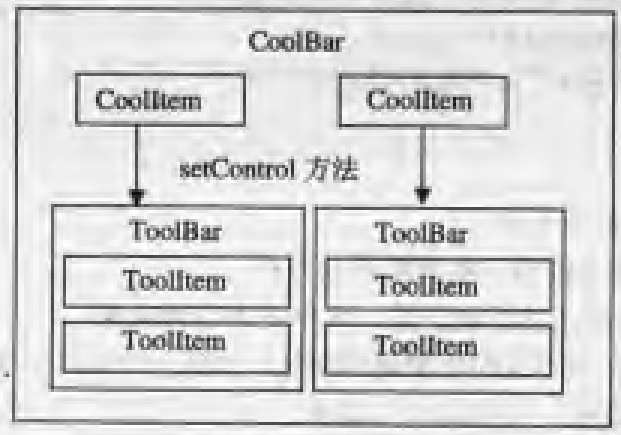


图 11.6　关系示意图

从图 11.6 可以看出 ToolBar 是建立在 CoolBar 容器里的，且 CoolItem 并非用来显示按钮，而是用来帮助 CoolBar 控制 ToolBar 的。回顾 9.3 节的图 9.6，比较可以发现 CoolBar、

CoolItem 和 ToolBar 三者正好与 TabFolder、TabItem 和 Group 三者是一样的关系。

实现图 11.4 效果的程序代码如下：

```
//--------文件名：SimpleEditor2.java--------
public class SimpleEditor2 {
    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(500, 375);
        shell.setText("SWT Application");
        //-------------------------
        shell.setLayout(new FillLayout());
        //用 ViewForm 做底层容器，并设为充满型布局
        final ViewForm viewForm = new ViewForm(shell, SWT.NONE);
        viewForm.setLayout(new FillLayout());
        //在 ViewForm 上建立 CoolBar 和文本框
        CoolBar coolBar = new CoolBar(viewForm, SWT.NONE);
        Text text = new Text(viewForm, SWT.BORDER | SWT.V_SCROLL);
        //设置 CoolBar 和文本框在 ViewForm 上的位置
        viewForm.setContent(text);
        viewForm.setTopLeft(coolBar);
        /*
         * 为 CoolBar 添加两条子工具栏
         */
        {
            ToolBar toolBar = new ToolBar(coolBar, SWT.NONE);
            /*
             * 为子工具栏 1 添加两个按钮
             */
            ToolItem toolItem = new ToolItem(toolBar, SWT.PUSH);
            toolItem.setText("取得");
            toolItem.setImage(new Image(display, "icons/selectAll.gif"));
            ToolItem toolItem2 = new ToolItem(toolBar, SWT.PUSH);
            toolItem2.setText("清除");
            toolItem2.setImage(new Image(display, "icons/remove.gif"));
            /*
             * 创建一个 coolItem 来控制此子工具栏
             */
            CoolItem coolItem = new CoolItem(coolBar, SWT.NONE);
            coolItem.setControl(toolBar);
            /*
             * 调整 toolBar 为合适的尺寸，并用 toolBar 的尺寸来调整 coolItem
             */
            toolBar.pack();//调整 toolBar 尺寸
            Point point=new Point(toolBar.getSize().x, toolBar.getSize().y);
            coolItem.setSize(point);//设定 coolItem 尺寸
```

```
            //设定 coolItem 的最小尺寸。如不设定，则按钮会被其他子工具栏遮住
            coolItem.setMinimumSize(point);
        }
        //以同样方式再创建一个子工具栏 2
            ToolBar toolBar = new ToolBar(coolBar, SWT.NONE);
            CoolItem coolItem = new CoolItem(coolBar, SWT.NONE);
            coolItem.setControl(toolBar);

            ToolItem toolItem = new ToolItem(toolBar, SWT.PUSH);
            toolItem.setImage(new Image(display, "icons/next.gif"));

            toolBar.pack();
            Point point=new Point(toolBar.getSize().x, toolBar.getSize().y);
            coolItem.setSize(point);
            coolItem.setMinimumSize(point);
        }
        //---------------------------------
        shell.layout();
        /*
         * （1）监听 coolBar 中子工具栏的改变，如有改变则重排 viewForm 中组件的位置。
         * 不加这个监听器，则当子工具栏由一行拖拉变成二行时，文本框将被遮住一部分。
         * （2）如果此监听器写在 shell.layout()前面，则窗口打开时，此事件会首先触发一
         * 次，这完全没有必要。由此可见，在 Eclipse 中监听器的代码位置常常是优化程序
         * 执行效率时要考虑和检查的地方
         */
        coolBar.addControlListener(new ControlAdapter() {
            public void controlResized(ControlEvent e) {
                viewForm.layout();
            }
        });
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) display.sleep();
        }
    }
}
```

11.3　菜单（Menu 类，MenuItem 类）

读者可以参照 ToolBar 和 ToolItem 来理解 Menu 和 MenuItem，本节以建立如图 11.7 所示的界面中的菜单为例。

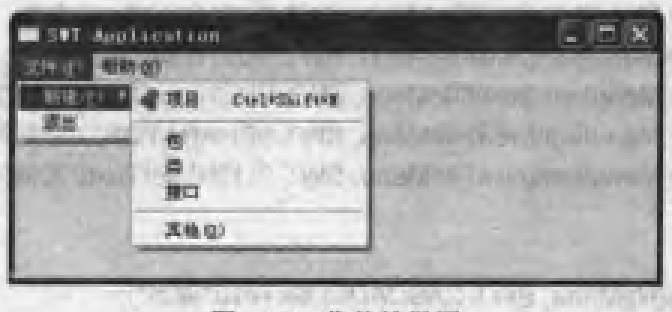


图 11.7　菜单效果图

实现代码如下：

```
//-------文件名：Menu1.java----------
//主菜单
Menu mainMenu = new Menu(shell, SWT.BAR);
shell.setMenuBar(mainMenu);
{
    // "文件" 项
    MenuItem fileItem = new MenuItem(mainMenu, SWT.CASCADE);
    fileItem.setText("文件(&F)");
    // "文件" 菜单
    Menu fileMenu = new Menu(shell, SWT.DROP_DOWN);
    fileItem.setMenu(fileMenu);
    {
        // "新建" 项
        MenuItem newFileItem = new MenuItem(fileMenu, SWT.CASCADE);
        newFileItem.setText("新建(&F)");
        // "新建" 菜单
        Menu newFileMenu = new Menu(shell, SWT.DROP_DOWN);
        newFileItem.setMenu(newFileMenu);
        {
            // "新建项目" 项
            MenuItem newProjectItem = new MenuItem(newFileMenu, SWT.PUSH);
            // "\t" 相当于间隔 Tab 键的空间
            newProjectItem.setText("项目\tCtrl+Shift+N");
            //定义快捷键 Shift+Ctrl+N
            newProjectItem.setAccelerator(SWT.CTRL + SWT.SHIFT + 'N');
            //设置菜单项的图标
            newProjectItem.setImage(new Image(null,"icons/project.gif"));
            //菜单项的单击事件
            newProjectItem.addSelectionListener(new SelectionAdapter() {
                public void widgetSelected(SelectionEvent e) {
                    MessageDialog.openInformation(null, "", "新建项目");
                }
            });
            //建立其他菜单项
            new MenuItem(newFileMenu, SWT.SEPARATOR);
```

```
                new MenuItem(newFileMenu, SWT.PUSH).setText("包");
                new MenuItem(newFileMenu, SWT.PUSH).setText("类");
                new MenuItem(newFileMenu, SWT.PUSH).setText("接口");
                new MenuItem(newFileMenu, SWT.SEPARATOR);//分隔符
                new MenuItem(newFileMenu, SWT.PUSH).setText("其他(&O)");
            }
        }
        new MenuItem(fileMenu, SWT.CASCADE).setText("退出");
    }
    MenuItem menuItem2 = new MenuItem(mainMenu, SWT.CASCADE);
    menuItem2.setText("帮助(&H)");
```

代码说明：

一项项的菜单让人有点头昏，所在阅读代码时一定要看清楚。

- 本例创建所有 Menu 对象的第一个参数都是 shell。
- 创建 MenuItem 对象的第一个参数是该 MenuItem 所在的 Menu 对象。
- 如果某 Menu 是某 MenuItem 的子菜单，则还要加一个设置：MenuItem.setMenu (Menu);。

Menu 的式样罗列如下：

- SWT.BAR，用于主菜单。
- SWT.DROP_DOWN，用于子菜单。
- SWT.POP_UP，用于右键菜单。

MenuItem 的式样罗列如下：

- SWT.CHECK，选择后前面会显示一个小勾。
- SWT.CASCADE，有子菜单的。
- SWT.PUSH，普通型。
- SWT.RADIO，选择后前面会显示一个圆点。
- SWT.SEPARATOR，分隔符。

右键弹出式菜单，效果如图 11.8 所示。

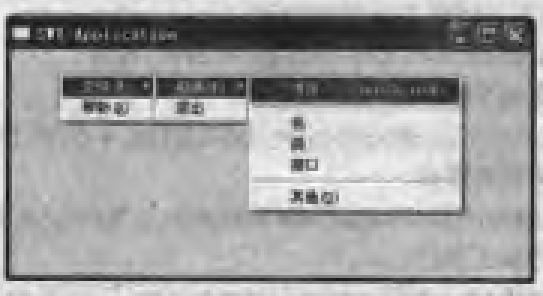


图 11.8　右键弹出式菜单效果图

实现的方法很简单，将程序开头的两句修改如下即可：

```
//主菜单
//Menu mainMenu = new Menu(shell, SWT.BAR);
//shell.setMenuBar(mainMenu);
```

```
Menu mainMenu = new Menu(shell, SWT.POP_UP);
shell.setMenu(mainMenu);
```

11.4　滑动条（Slider）、刻度条（Scale）、进度条（ProgressBar）

1．滑动条（Slider 类）

滑动条的外观如图 11.9 所示，在这个窗口里有两个组件，右边是一个滑动条，左边是一个用来显示滑动条当前值的文本框。

图 11.9　滑动条

图 11.9 的实现代码如下：

```
//--------文件名：Slider1.java---------
shell.setLayout(new RowLayout());
final Text text = new Text(shell, SWT.BORDER);
//创建一个滑动条，默认是水平型，用式样 SWT.VERTICAL 可变成垂直型
final Slider slider = new Slider(shell, SWT.NONE);
slider.setMinimum(0); //最小值
slider.setMaximum(100);//最大值
slider.setIncrement(5);//滚动间隔值
slider.setPageIncrement(10);//翻页间隔值
slider.setSelection(25);//初始值
/*
 * 监听滑动条的值的改变，并将值显示在文本框中
 */
slider.addSelectionListener(new SelectionAdapter() {
    pubiic void widgetSelected(SelectionEvent e) {
        text.setText("" + slider.getSelection());
    }
});
```

2．刻度条（Scale 类）

刻度条和滑动条在使用方法上是一样的。将滑动条的实现代码中，创建滑动条的语句修改如下，即可得到如图 11.10 所示的刻度条。

```
//final Slider slider = new Slider(shell, SWT.NONE);
final Scale slider = new Scale(shell, SWT.NONE);
```

3．进度条（ProgressBar 类）

进度条的效果如图 11.11 所示。

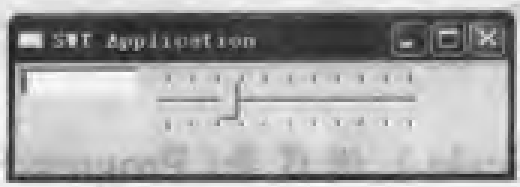


图 11.10　刻度条

图 11.11　进度条

进度条的使用方法和滑动条、刻度条类似，主要也是三种方法：setMinimum、setMaximum 分别定义最小值、最大值，setSelection 方法控制进度条的当前位置。具体实现代码如下：

```
//--------文件名：ProgressBar1.java---------
shell.setLayout(new RowLayout());
//创建一个滑动条。这里用平滑型式样 SWT.SMOOTH，默认是方格型
final ProgressBar progressBar = new ProgressBar(shell, SWT.SMOOTH);
progressBar.setMinimum(0); //最小值
progressBar.setMaximum(100);//最大值
//创建一个 GO 按钮
Button button = new Button(shell, SWT.BORDER);
button.setText("GO");
button.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        //每次循环 progressBar 前进一格
        for (int i = 1; i < 11; i++) {
            try {
                Thread.sleep(1000);
            } catch (Throwable e2) {} //间隔一秒
            progressBar.setSelection(i * 10);
        }
    }
});
```

程序说明：

- 这个程序演示了进度条的使用，但并不完美，当开始执行处理过程时，界面就很难再响应其他的操作。解决这个问题需要用到 SWT 线程的知识，请参阅 13.3 节。
- JFace 包中有一种 Dialog 组件叫做“ProgressMonitorDialog”，它也有表示任务执行进度的功能，后面章节将会介绍。

11.5　画布（Canvas 类）

Canvas 主要用于显示图像，既可以在 Canvas 上画图，也可以直接将图片显示于其中。本节通过实现如图 11.12 所示效果的一个小程序，来演示 Canvas 使用的三个主要方面：设置图像、更换图像、清除图像。

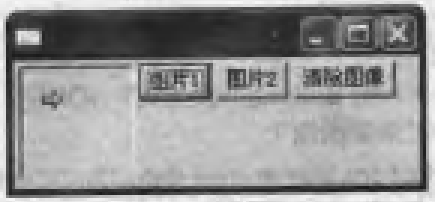


图 11.12　Canvas 效果图

实现代码如下：

```
//---------文件名：Canvas1.java-----------------
public class Canvas1 {
    static Image image;

    public static void main(String[] args) {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        shell.setSize(327, 253);
        shell.setText("SWT Application");
        //-----------------新插入的界面核心代码----------------------
        shell.setLayout(new RowLayout());
        final Canvas canvas = new Canvas(shell, SWT.BORDER);
        //监听 canvas 的重绘事件。
        canvas.addPaintListener(new PaintListener() {
            public void paintControl(final PaintEvent event) {
                //将其图像显示在 canvas 上，（10,10）是显示图像的左上角坐标
                if (image != null)
                    event.gc.drawImage(image, 10, 10);
            }
        });

        Button button1 = new Button(shell, SWT.NONE);
        button1.setText("图片 1");
        button1.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                image = new Image(display, "icons/next.gif");
                canvas.redraw();
            }
        });

        Button button2 = new Button(shell, SWT.NONE);
        button2.setText("图片 2");
        button2.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                image = new Image(display, "icons/refresh.gif");
                canvas.redraw();
            }
```

```
        });

        Button clearButton = new Button(shell, SWT.NONE);
        clearButton.setText("清除图像");
        clearButton.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                if (image != null) {
                  image.dispose();
                  image = null;
                  canvas.redraw();
                }
            }
        });
    //----------------END----------------------------------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
              display.sleep();
        }
    }
}
```

11.6 表格（Table 类）

表格效果如图 11.13 所示，本实例将实现此界面中的如下功能：

- 单击“新增”按钮向表格中增加记录。
- 表格可以多选。
- 能够删除所选的记录。
- 双击表格弹出显示记录值的提示框。

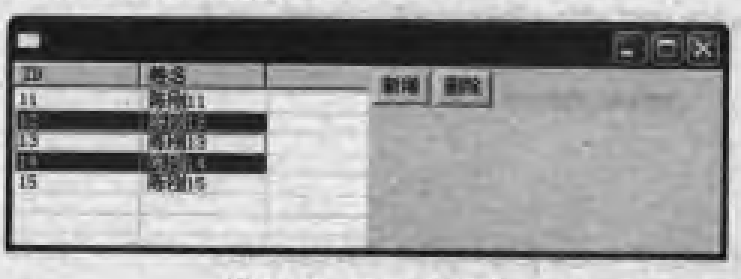


图 11.13　Table 效果图

前面的章节介绍过了选项卡的 TabFolder、TabItem，菜单的 Menu、MenuItem，工具栏的 ToolBar、ToolItem，同样 Table 也有自己的 Item：TableItem。TableItem 代表表格中的一条记录。不过在实际项目中，特别是用到数据库的项目，一般都不用 Table，而用 JFace 中基于 Table 扩展的 TableViewer 来作为表格组件（以后将会介绍 TableViewer 的使用）。

本实例的具体实现代码如下：

```
//---------文件名：Table1.java----------
shell.setLayout(new FillLayout());
/*
 * 创建一个 Table 对象，在式样里设置它:可多选、全列选择。
 * 并用两条语句设置它显示表头和表格线
 */
final Table table = new Table(shell, SWT.MULTI | SWT.FULL_SELECTION);
table.setHeaderVisible(true);
table.setLinesVisible(true);
/*
 * 为 Table 增加两列并设置列宽都是 80
 */
TableColumn col1 = new TableColumn(table, SWT.NONE);
col1.setText("ID");
col1.setWidth(80);
TableColumn col2 = new TableColumn(table, SWT.NONE);
col2.setText("姓名");
col2.setWidth(80);
/*
 * 创建一个 Composite 容器，并放入两个按钮
 */
Composite composite = new Composite(shell, SWT.NONE);
composite.setLayout(new RowLayout());
Button addButton = new Button(composite, SWT.NONE);
addButton.setText("新增");
addButton.addSelectionListener(new SelectionAdapter() {
    int i = 1;
    public void widgetSelected(SelectionEvent e) {
        TableItem item = new TableItem(table, 0); //创建 Item
        item.setText(new String[] { "" + i, "陈刚" + i });//给 Item 设值
        i++;
    }
});
Button removeButton = new Button(composite, SWT.NONE);
removeButton.setText("删除");
removeButton.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        //得到装有所有被选择记录的序号的数组
        int[] selInices = table.getSelectionIndices();
        //Table 根据数组中的序号将 Item 删除
        table.remove(selInices);
    }
});
/**
```

```
 * 监听 Table 的鼠标双击事件
 */
table.addMouseListener(new MouseListener() {
    public void mouseDoubleClick(MouseEvent e) {
        int selIndex = table.getSelectionIndex();
        TableItem item = table.getItem(selIndex);
        String str = "列 1 = " + item.getText(0) + "\n 列 2 = " + item.getText(1);
        MessageDialog.openInformation(null, null, str);
    }
    public void mouseDown(MouseEvent e) {}
    public void mouseUp(MouseEvent e) {}
});
```

11.7　树（Tree 类）

树的效果如图 11.14 所示，本节的实例将实现此界面效果。

图 11.14　Tree 效果图

和 Table 一样，Tree 也有自己的 Item：TreeItem，由它表示树中的每一个结点。很多方法 Table 和 Tree 都是一样的，大家可以参照着理解，而且 JFace 也有一个基于 Tree 扩展的 TreeViewer，实际用到数据库的项目一般都用它。

通过本实例，读者将学习到：

- 如何创建一棵树。
- 当单击树结点时如何得到树结点的值。

实例具体代码如下：

```
//--------文件名:Tree1.java----------
shell.setLayout(new FillLayout());
/*
 * 创建一个 Table 对象，在式样里设置它可多选、全列选择。
 * 并用两条语句设置它显示表头和表格线
 */
final Tree tree = new Tree(shell, SWT.SINGLE);
TreeItem chinaItem = new TreeItem(tree, SWT.NONE);
chinaItem.setText("中国");
    new TreeItem(chinaItem, SWT.NULL).setText("北京");
    new TreeItem(chinaItem, SWT.NULL).setText("台湾");
```

```
    new TreeItem(chinaItem, SWT.NULL).setText("桂林");
new TreeItem(tree, SWT.NONE).setText("美国");
/**
 * 当单击 Tree 的结点时，结点文字显示于窗口标题栏中
 */
tree.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        /*
         * Tree 也支持多选，所以 getSelection 方法返回的是一个 Item 数组。
         * 但本例没有设 Tree 为多选式样，而且单击一次也只能有一个结点，
         * 所以只取数组中第一个 Item 的文字即可
         */
        TreeItem[] items = tree.getSelection();
        shell.setText(items[0].getText());
    }
});
```

11.8　表格型树（TableTree 类）

表格型树的效果如图 11.15 所示，它是表格和树的结合体，其整体是一个表格，然后可以在表格的单元格中显示树状的数据。

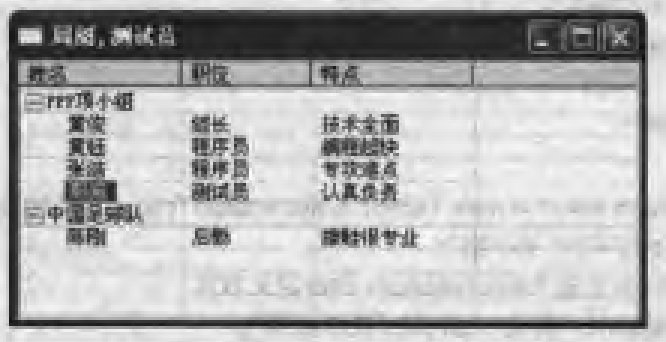


图 11.15　TableTree 效果图

要完成本节实例，需要两个类：TableTree、TableTreeItem，其中 TableTreeItem 是代表 TableTree 的一条记录，它可以嵌套生成其子结点项。所以，虽然 TableTree 里面有树的界面，但程序中并不需要用到 Tree 组件，TableTree 中的每一行都是一个 TableTreeItem 对象。

通过本实例读者将学习到：

- 如何创建 TableTree，并向其中输入数据。
- 如何得到选择的记录的值。

实例的具体代码如下：

```
//--------文件名：TableTree1.java---------
public class TableTree1{
    public static void main(String[] args) {
```

```
final Display display = Display.getDefault();
final Shell shell = new Shell();
shell.setSize(500, 375);
//----------------------------------------
shell.setLayout(new FillLayout());
//创建一个 TableTree
final TableTree tableTree = new TableTree(shell, SWT.BORDER);
/*
 * 显示表头和表格线
 */
Table table = tableTree.getTable();
table.setHeaderVisible(true);
table.setLinesVisible(true);
/*
 * 加入三列
 */
TableColumn col1 = new TableColumn(table, SWT.NONE);
col1.setText("姓名");
col1.setWidth(100);
TableColumn col2 = new TableColumn(table, SWT.NONE);
col2.setText("职位");
col2.setWidth(80);
TableColumn col3 = new TableColumn(table, SWT.NONE);
col3.setText("特点");
col3.setWidth(100);
/*
 * 第一个树
 */
TableTreeItem item1 = new TableTreeItem(tableTree, SWT.NONE);
item1.setText("PPP 项小组");
//createItem 为生成 TableTreeItem 的自定义方法
createItem(item1, "黄俊", "组长", "技术全面");
createItem(item1, "黄钰", "程序员", "编程超快");
createItem(item1, "张浩", "程序员", "专攻难点");
createItem(item1, "周阔", "测试员", "认真负责");
/*
 * 第二个树
 */
TableTreeItem item2 = new TableTreeItem(tableTree, SWT.NONE);
item2.setText("中国足球队");
createItem(item2, "陈刚", "后勤", "擦鞋很专业");
/*
 * 监听选择事件，并将所选项的记录文字显示在窗口标题栏上
 */
tableTree.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
```

```
                TableTreeItem[] ttiArray = tableTree.getSelection();
                TableTreeItem tti = ttiArray[0];
                String str = tti.getText(0) + "," + tti.getText(1);
                shell.setText(str);
            }
        });
        //-----------------------------------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) display.sleep();
        }
    }
    /**
     * 自定义方法，生成 TableTreeItem
     */
    static TableTreeItem createItem(TableTreeItem parentItem,
                                    String str1,
                                    String str2,
                                    String str3) {
        TableTreeItem item2 = new TableTreeItem(parentItem, SWT.NONE);
        item2.setText(str1);
        item2.setText(1, str2);
        item2.setText(2, str3);
        return item2;
    }
}
```

11.9　本 章 小 结

本章介绍的组件在创建应用程序时，将会经常用到，其中 Table、Tree、TableTree 是 JFace 中 TableViewer、TreeViewer、TableTreeViewer 三组件的基础。如果开发 Eclipse 插件(plugin)，工具栏、菜单的组件很少用到，一般都是扩展 Eclipse 本身的工具栏、菜单。

第12章 图 像

12.1 图像（Image 类）

开发项目少不了用到图像，在 SWT 中图像是 Image 类，它属于 org.eclipse.swt.graphics 包。Image 有多种构造函数，本书选最常用的 new Image (Device device, String filename)来介绍。其中，参数 Device 称为设备，它包含 Display（显示）和 Printer（打印）两种设备，如图 12.1 所示。如果要在屏幕上显示，当然是用 Display 显示设备。

图 12.1 Device 谱系图

以下实例代码是使用“绝对路径”来得到一个图像，并设置图像背景为红色，然后显示在一个按钮上。

```
button.setText("button");
Image image = new Image(display, "C:\\refresh.gif");
//以上一句也可以写成：Image image = new Image(null, "c:/refresh.gif");
image.setBackground(Display.getCurrent().getSystemColor(SWT.COLOR_RED));
button.setImage(image);
```

程序说明：

- new Image(null, "c:/refresh.gif")，它的第一个参数为 null，这时会默认使用当前的 Display 来做显示设备。
- 如果在所指路径上没有图片，会弹出错误提示。

实例的效果如图 12.2 所示。

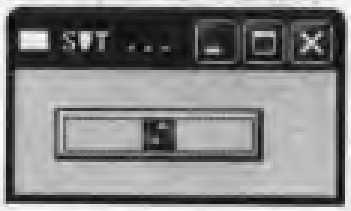


图 12.2 运行效果图

因为无法知道用户会将我们的软件安装在什么目录下，所以用绝对路径是会出问题的，应该用相对路径。代码如下：

```
button.setImage(new Image(display, "icons/refresh.gif"));
```

这里要注意相对路径的起始点是项目主目录。例如我们这个项目名叫 myswt，那么相对路径就是从 myswt 目录开始，在本书就是对应于 C:\eclipse\workspace\myswt\icons\refresh.gif。

如果开发插件，由于插件是基于 Eclipse 环境运行的，所以还可以利用 Eclipse 自带的图像。使用方法如下：

```
ISharedImages sharedImages = PlatformUI.getWorkbench().getSharedImages();
Image image = sharedImages.getImage(ISharedImages.IMG_OBJS_WARN_TSK);
```

12.2 图像（Image 类）存在的问题

Image 是比较重量级的对象，占用内存很大，一般来说，当应用程序退出时会释放掉 Image 所占资源。但某些操作系统由于特殊原因，Image 仍有可能继续存在于系统内存中，而未被释放。并且有些操作系统对同一时间内可拥有的图像数量有着严格限制，如果超过限制将有可能导致系统崩溃。笔者过去在 Eclipse 2.0 下使用 SWT Designer 进行界面设计时，界面上组件一多，就很容易因为内存耗尽而导致异常退出，当时，这是 SWT Designer 的一个严重的不足。

由于 Image 的代价昂贵，为了提高 Image 对象的利用率，在 Eclipse 中 Image 是可以被各组件共享的。当窗口关闭时，Image 将不会被自动清除掉，而是继续保留在系统中，以便被其他窗口组件使用。因此如果肯定程序中不会再使用某个 Image 对象，这时就应该用 Image 的 dispose()方法，立即释放掉 Image。

像 Image 这种情况的对象还有 Font（字体）、Color（颜色）等。

12.3 图像描述符（ImageDescriptor 类）

由于 Image 存在的一些问题，JFace 包提供了一个轻量级的 ImageDescriptor 类，它属于 org.eclipse.jface.resource 包。ImageDescriptor 并不存储图像本身，而是在程序需要时才会创建图像，并且有些界面组件也只接受 ImageDescriptor 做参数来设置图像。

ImageDescriptor 是一个抽象类（abstract），所以它不能用 new 的方式来创建。在这里介绍用 ImageDescriptor 的两种静态方法 createFromFile、createFromURL 来生成图像描述符的方法。

1. createFromFile 方法

```
ImageDescriptor imageDesc=ImageDescriptor.createFromFile(Abc.class ,"icons/refresh.gif");
```

此句指当要创建图像时，会通过 Abc.class 类所在目录下的 icons 子目录来加载 refresh.gif，也就是 icons 相对于 Abc.class 所在目录来定位。此时，imageDesc 只是包含了指

向 refresh.gif 的信息，还没有真正加载 refresh.gif。Abc.class 这个参数可以是项目中的其他类，甚至是写这个语句的类也可以。

注意：应该将 icons 图像目录复制到 Abc.java 所在的目录下，src 和 bin 两目录结构会保持同步，Eclipse 将自动把 icons 复制到 Abc.class 的目录之下。

2. createFromURL 方法

- 创建一个图像描述符，图像定位在 C:\icons\refresh.gif 路径。实现代码如下：

```
URL url = new URL("file:\\C:\\icons\\refresh.gif");
ImageDescriptor imageDesc=ImageDescriptor.createFromURL(url);
```

- 创建一个图像描述符，图像定位在项目主目录（myswt）下的子目录 icons 中的 refresh.gif，在本书即是 C:\eclipse\workspace\myswt\icons\ refresh.gif。实现代码如下：

```
URL url = new URL("file:icons/refresh.gif");
ImageDescriptor imageDesc=ImageDescriptor.createFromURL(url);
```

说明：创建 URL 的语句会抛出一个异常，它需要用 try…catch 包住。

3. 利用 Eclipse 自带的图像

如果开发插件，由于插件是基于 Eclipse 环境运行的，所以还可以利用 Eclipse 自带的图像。使用方法如下：

```
ImageDescriptor imageDesc = WorkbenchImages.getImageDescriptor(
                                IWorkbenchGraphicConstants.IMG_ETOOL_HOME_NAV);
```

4. ImageDescriptor 总结

ImageDescriptor 比 Image 有个好处：如果前者所指图像路径错误，则会自动用一个红色方块图像来代替，而不会像后者那样弹出错误提示框。另外，由 ImageDescriptor 也可得到 Image，方法是：Image image= imageDesc.createImage();。

12.4 图像注册表（ImageRegistry 类）

JFace 提供了一个能高速缓存图像和图像描述符的图像注册表。它的设计思路是：先创建指向图像的图像描述符，然后将图像描述符加上一个键值添加到图像注册表中，这个键值是和图像描述符一一对应的。当希望获得图像时，就可以用键值从注册表中取得，如图 12.3 所示。

当清除顶级显示时，在 Eclipse 中会将图像注册表中注册过的图像释放掉。还可以建立多个这样的图像注册表，以方便更频繁而且细致地释放图像资源。这里给出一些图像注册表加值、取值、移除的方法。其具体代码如下：

```
/*
 * 创建一个图像注册表。因为程序需要共享图像注册表 ImageRegistry，所以在实际项目中
```

```
* 最好将 ImageRegistry 设置成静态变量，以使它可以被全局访问
*/
ImageRegistry imageRegistry = new ImageRegistry();
//创建一个图像描述符对象
URL url = new URL("file:icons/refresh.gif");
ImageDescriptor imageDesc=ImageDescriptor.createFromURL(url);
//将图像描述符(ImageDescriptor)添加到图像注册表，键值为 "refresh"
imageRegistry.put("refresh", imageDesc);
//将图像从图像注册表中取出
Image image=imageRegistry.get("refresh");
//从图像注册表中取出图像描述符
ImageDescriptor imageDesc2=imageRegistry.getDescriptor("refresh");
/*
* 将图像从注册表里移除。如果有组件正在使用这个图像，图像依然会被释放掉，组件上就不会
* 显示此图像。另外，如果注册表里没有这个键值"refresh"，也不会产生错误或异常
*/
imageRegistry.remove("refresh");
```

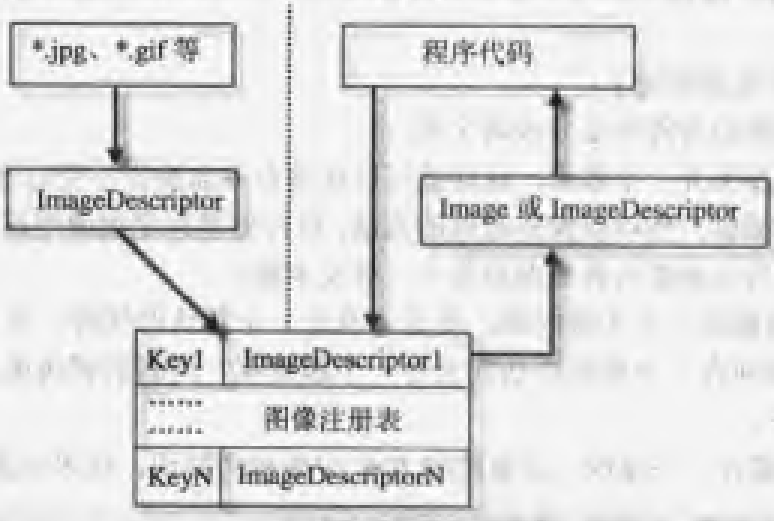


图 12.3　图像注册表的设计思路示意图

12.5　本 章 小 结

本章详细介绍了图像的使用知识，它主要分为三大内容：图像、图像描述符和图像注册表，这三个内容是互相依赖，一步步发展出来的。其中图像注册表至关重要，在项目开发的初期，首先就是要创建一个图像注册表，至于要创建一个还是多个，这要看项目的大小，一般来说一个就足够了。在第 22 章中，将会详细演示如何在实际项目开发中使用图像、图像描述符和图像注册表。

第 13 章　SWT 的线程

13.1　SWT 线程简介

大部分情况下，GUI 界面编程都不用考虑线程问题，SWT 已经帮助我们隐藏了底层的线程调用。但一些特殊应用的实现，却不得不涉及 SWT 线程编程。例如，11.4 节中进度条的实例，它不是很完善，当开始执行后界面很难被再操作，有一种死机的感觉，这主要是因为其事件处理占用了整个线程。

在实际项目开发中，一种常见的应用是：单击界面的“执行”按钮后，在后台处理一些超长时间的任务，任务的执行情况要及时反映在界面上，而且还不能影响前台界面的其他操作。

这种应用的实现思路如下：

- 前台界面和后台程序是分成两个类。
- 为后台程序另开一个线程，这样就可以让前台界面操作不受后台处理的影响。
- 前台界面提供一些可以操作组件的方法，后台处理程序则调用这些方法将执行情况的字符串写入到前台界面的组件中（如文本框）。

这种思路必须解决一个关键问题：界面本身有一个默认的线程，后台程序又是另外一个线程，那么该如何在一个线程中访问另外一个线程呢？问题的解决很简单，仅要记住的两个核心要点如下：

（1）为后台新开一个线程。这要用到 JDK 中线程的知识，代码示意如下：

```
new Thread(){//新开一个线程，这是匿名内部类的写法
    public void run(){
        //开始后台任务……
    }
}.start();//start 表示立即启动线程
```

（2）从后台处理线程中访问前台界面组件，关键在于使用 Display 对象，因为 Display 主要负责管理事件循环和控制 UI 线程和其他线程之间的通信。Display 的具体使用方法如下：

```
Display.asyncExec(new Runnable(){//这又是一个匿名内部类的写法
    public void run(){
        //对前台界面的操作……
    }
});
```

在这里 Display 对象很关键，因此顺便给出得到 Display 对象的两种方法：

- Display.getDefault()，得到一个默认的 Display 对象，应用程序一般只需要一个 display 对象。
- Display.getCurrent ()，得到当前运行线程所在的 Display 对象，如果生成了两个以上的 Display 对象，则可以用这个方法。

本节对 SWT 线程应用作了简介和分析，下面将给出具体应用的实际代码。

13.2　一个 SWT 线程的实例

本节将实现一个应用了 SWT 线程知识的典型实例，实例的界面如图 13.1 所示。

图 13.1　实例界面

本实例分两个类：第一个是前台界面类，第二个是后台处理类。实例功能说明如下：

- 单击“执行”按钮时，开始后台处理。
- 单击“停止”按钮时，立即中断后台处理。
- 后台处理的任务数根据文本框的值来决定。
- 进度条将实时反映后台处理的进度。
- 底部的文本框将以文字形式反映后台处理的进度情况。

实例的具体代码如下：

1．前台界面类

```
public class TaskGUI {
    private Display display = Display.getDefault();
    private Shell shell = new Shell();
    private Task task = new Task(this); //Task 为后台处理类
    //将界面组件设为类的实例变量
    private Text taskCountText; //任务数文本框
    private Button startButton;// “执行” 按钮
    private Button stopButton;// “停止” 按钮
    private ProgressBar progressBar;//进度条
```

```
private Text consoleText;//输出调试信息的文本框

public static void main(String[] args) {
    try {
        TaskGUI window = new TaskGUI();
        window.open();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void open() {
    shell.setSize(300, 300);
    shell.setLayout(new GridLayout());
    //任务数文本框和按钮的组
    Group group = new Group(shell, SWT.NONE);
    group.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    group.setLayout(new GridLayout(4, false));
    //任务数文本框
    new Label(group, SWT.NONE).setText("任务数：");
    taskCountText = new Text(group, SWT.BORDER);
    taskCountText.setText("100");//默认任务数 100
    taskCountText.setLayoutData(new GridData(100, -1));
    taskCountText.addVerifyListener(new VerifyListener() {
        public void verifyText(VerifyEvent e) {//只能输入数值
            e.doit = "0123456789".indexOf(e.text) >= 0;
        }
    });
    //执行按钮
    startButton = new Button(group, SWT.PUSH);
    startButton.setText("执行");
    startButton.addSelectionListener(new SelectionAdapter() {
        public void widgetSelected(SelectionEvent e) {
            //设置两按钮的“置灰、使能”状态
            setButtonState(false);
            //得到任务数，多线程使用的变量要求类型为 final
            String str = taskCountText.getText();
            final int taskCount = new Integer(str).intValue();
            //设置进度条的格数
            progressBar.setMaximum(taskCount - 1);
            consoleText.insert("后台处理线程开始启动.....\n");
            //为后台任务新开一个线程
            new Thread() {
                public void run() {
                    task.start(taskCount);
                }
```

```
                }.start();
                consoleText.insert("后台处理线程启动结束\n");
            }
        });
        //停止按钮
        stopButton = new Button(group, SWT.PUSH);
        stopButton.setText("停止");
        stopButton.setEnabled(false);
        stopButton.addSelectionListener(new SelectionAdapter() {
            public void widgetSelected(SelectionEvent e) {
                task.stop();//后台任务停止
            }
        });
        //进度条
        progressBar = new ProgressBar(shell, SWT.NONE);
        progressBar.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        //输出调试信息的文本框
        consoleText = new Text(shell, SWT.MULTI | SWT.BORDER | SWT.V_SCROLL);
        consoleText.setLayoutData(new GridData(GridData.FILL_BOTH));
        //----------------------------------------
        shell.layout();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }

    /**
     * 使用“执行”“停止”两按钮的“置灰/使能”状态
     */
    public void setButtonState(boolean bFlag) {
        startButton.setEnabled(bFlag);
        stopButton.setEnabled(!bFlag);
    }

    /**
     * 为后台类取界面组件的几个 get 方法
     */
    public Shell getShell() {
        return shell;
    }

    public Text getConsoleText() {
        return consoleText;
    }
```

```
    public ProgressBar getProgressBar() {
        return progressBar;
    }
}
```

2. 后台任务类

```
public class Task {
    private TaskGUI gui; //得到前台界面对象
    private boolean stopFlag; //是否停止的标识

    /**
     * 构造函数，取得前台界面对象
     */
    public Task(TaskGUI taskGUI) {
        this.gui = taskGUI;
    }

    /**
     * 停止执行
     */
    public void stop() {
        stopFlag = true;
    }

    /**
     * 开始执行
     */
    public void start(int taskCount) {
        stopFlag = false; //将执行状态初始化成执行
        intsertConsoleText("后台线程开始执行任务.....\n");
        for (int i = 0; i < taskCount; i++) {
            //一旦发现执行状态为停止，则退出此循环
            if (stopFlag)
                break;
            //每隔 0.1 秒一次循环，用于模拟表示一个需时较长的任务
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            intsertConsoleText("任务" + (i + 1) + "处理完毕\n");
            //移动进度条的进度
            moveProgressBar(i);
        }
```

```
        intsertConsoleText("后台线程结束执行任务!!!!!!\n");
        //修改界面按钮状态
        setTaskGUIButtonState(true);
    }

    /**
     * 界面按钮状态
     */
    private void setTaskGUIButtonState(final boolean bFlag) {
        Display.getDefault().asyncExec(new Runnable() {
            public void run() {
                gui.setButtonState(bFlag);
            }
        });
    }

    /**
     * 进度条显示
     */
    private void moveProgressBar(final int progress) {
        Display.getDefault().syncExec(new Runnable() {
            public void run() {
                gui.getProgressBar().setSelection(progress);
            }
        });
    }

    /**
     * 插入文本
     */
    private void intsertConsoleText(final String str) {
        Display.getDefault().syncExec(new Runnable() {
            public void run() {
                gui.getConsoleText().insert(str);
            }
        });
    }
}
```

程序说明：

- 两个类的代码虽然很长，但关键代码还是 13.1 节所说的两个核心要点的使用。
- 从代码的调试信息情况可知，当为后台处理新开一个线程（new Thread）时，界面的处理流程没有在“task.start(taskCount);”处停顿，而是继承执行并输出了“后台处理线程启动结束”字符串。

13.3　对 11.4 节进度条实例的改进

本节将用 SWT 线程来修复 11.4 节进度条实例中不完善的地方，最后完成的代码如下。

1．11.4 节的旧程序

```
public void widgetSelected(SelectionEvent e) {
    //每次循环 progressBar 前进一格
    for (int i = 1; i < 11; i++) {
        try {
            Thread.sleep(1000);
        } catch (Throwable e2) {} //间隔一秒
        progressBar.setSelection(i * 10);
    }
}
```

2．第一种修改方式（文件名：ProgressBar2.java）

```
public void widgetSelected(SelectionEvent e) {
    new Thread() {
        public void run() {
            //每次循环 progressBar 前进一格
            for (final int[] i = { 1 }; i[0] < 11; i[0]++) {
                try {
                    Thread.sleep(1000);
                } catch (Throwable e2) {} //间隔一秒

                display.asyncExec(new Runnable() {
                    public void run() {
                        progressBar.setSelection(i[0] * 10);
                    }
                });
            }
        }
    }.start();
}
```

程序说明：

之所以要将循环变量写成数组的方式，是因为 display.asyncExec 外的变量要被使用必须是 final 型。如果仅仅将循环变量 i 变为 for (final i=1 …)是不行的，因为 final 型的变量不允许再赋值，在 i++处会报错。

用数组方式是利用了 final 型数组的一个特性：数组整体不能再赋值，但数组元素可以再赋值。这种修改方式的缺陷是变量 i 的用法太过怪异，不利于今后代码的维护。

3. 第二种修改方式（文件名：ProgressBar3.java）

```
public void widgetSelected(SelectionEvent e) {
    new Thread() {
        public void run() {
            //每次循环 progressBar 前进一格
            for (int i = 1; i < 11; i++) {
                try {
                    Thread.sleep(1000);
                } catch (Throwable e2) {} //间隔一秒

                final Integer i_copy = new Integer(i);
                display.asyncExec(new Runnable() {
                    public void run() {
                        progressBar.setSelection(i_copy.intValue() * 10);
                    }
                });
            }
        }
    }.start();
}
```

程序说明：

这种方式可以不改变循环变量 i 的定义，而是用 final 型的 Integer 对象作为中介。但这种方式有一个小小的缺陷：每一次循环都会产生一个新的 Integer 对象。虽然 Integer 对象耗用的资源很微小，但如果你的程序对执行效率要求极端严格，那么还是用第一种修改方式较好。

13.4　本 章 小 结

本章介绍了 SWT 线程中的核心知识，并给出了两个实际应用的例子。SWT 线程虽然实现较简单，但其涉及的 Java 多线程编程却是相当复杂的，特别要注意多线程对变量的影响，具体内容请参阅 Java 书籍中关于线程的内容。一般情况下，界面开发中尽量不要去涉及多线程，即使要用也要十分小心，不可随便大意，要知道多线程引起的 BUG 是很难查找出来的。

第 14 章　表格（TableViewer 类）

14.1　概　　述

JFace 是 SWT 的扩展，它提供了一组功能强大的界面组件，其中包含表格、树、列表、对话框、向导对话框等，从本章之后就开始专门来介绍这些 JFace 组件。

表格是一种在软件系统很常见的数据表现形式，特别是基于数据库的应用系统，表格更是不可缺少的界面组件。SWT 的表格组件（Table 类）前面已经介绍过了，但在实际项目开发中一般还是用 JFace 的表格组件 TableViewer。TableViewer 组件是在 SWT 的 Table 组件基础上扩展而来的，但 Table 并非 TableViewer 的父类，从图 14.1 两个类的谱系图就可以看出这两个类甚至都不属于同一个族系。

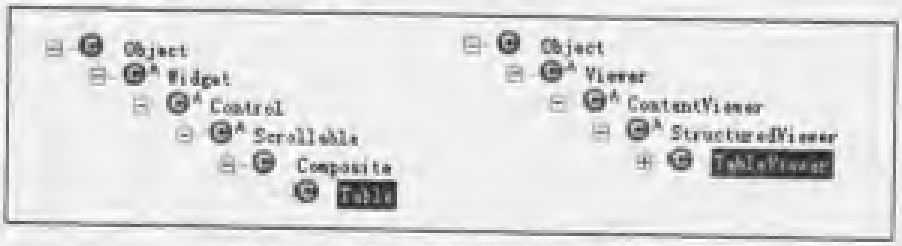


图 14.1　Table、TableViewer 的谱系图

从下面的 TableViewer 类源代码可以看到，TableViewer 把 Table 作为一个实例变量，从而实现了对 Table 功能的扩展。

```
public class TableViewer extends StructuredViewer {
    private TableViewerImpl tableViewerImpl;
    private Table table; //把 Table 类作为一个实例变量
    private TableEditor tableEditor;
    ......
}
```

本章就如何使用表格组件 TableViewer 类来展开讲解，并通过一步步地创建一个完整的表格应用实例来串起表格的知识点，实例的最后界面如图 14.2 所示。

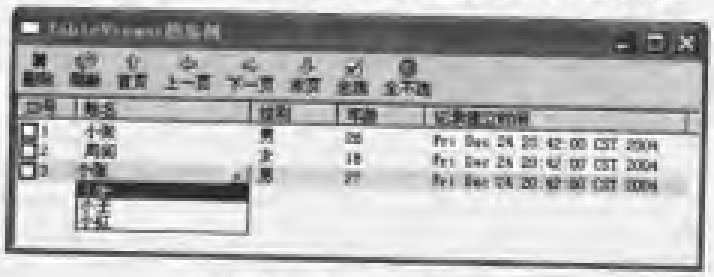


图 14.2　实例的最后界面

14.2　让数据在 TableViewer 中显示出来

本节内容将演示如何把数据记录用 TableViewer 显示出来，实例运行效果如图 14.3 所示。

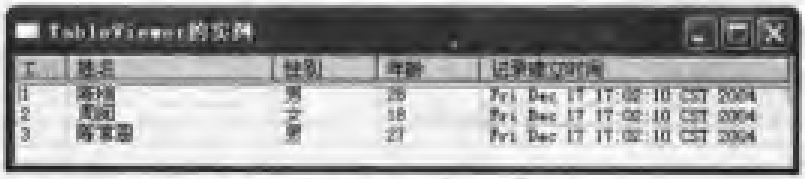


图 14.3　TableViewer 效果图

14.2.1　实例的数据模型介绍

本实例用 TableViewer 来显示一个数据表中的三条记录，每一条记录对应某一个人的基本资料，记录有五个字段：ID 号（数值型）、姓名（字符型）、性别（布尔型）、年龄（数值型）、记录建立时间（日期型）。

如何在程序中体现和操作这些数据记录呢？在过去，像 ASP、PHP 这类面向过程的编程模式，我们习惯了这样操作数据：从数据库中读取数据，也不用对数据做任何封装，直接将数据一条条地显示在表格中。

现在用 Java 这种面向对象的编程语言，我们应该用更规范的方式来操作数据：将数据库中的记录当做一个数据对象，用一个类来表示它，数据表的字段写成类的实例变量，这样的类在 Java 中叫做实体类。EJB 的数据操作方式就是这样的，还有现在风头正健的 Hibernate 也是用这种方式。

数据库与表格显示之间加上了实体类，如此一来，以前的“数据表→表格显示”方式就分成了两个步骤“数据库→实体类→表格显示”。有些习惯了以前编程方式的人也许会觉得多了一个步骤太麻烦，但这种方式是很有好处的：

- 表格显示的代码不再和数据库相关。例如，将数据库由 Oracle 移植到 MySQL 时就不需要更改“数据库→实体类”这个环节的代码。
- 零散的字段变量统一在一个类中，程序代码结构更紧凑、清晰，有利于今后代码的维护。不要小看维护问题，很多系统做好后不敢再改，害怕改动后会牵涉到其他地方，其中原因之一就是代码结构太乱、编程不规范所致。
- 将数据封装在一个实体类中，在数据传递时也方便多了，可以将实体类作为一个参数在方法与方法之间来回传递。

14.2.2　创建数据表的实体类

我们依照表中的字段来创建一个相应的实体类，类名为 PeopleEntity，代码如下所示。

```
//---------- 文件名：PeopleEntity.java ----------
/**
 * 本类包含五个不同数据类型的变量，分别对应数据库中的五个字段。变量为 private，即只能由
 * 类的内部代码访问，外界只能通过这些变量相应的 set/get 方法来访问这些变量
 */
public class PeopleEntity {
    private Long id; //惟一识别码，在数据库里常为自动递增的 ID 列
    private String name; //姓名
    private boolean sex; //性别 true 男，flase 女
    private int age; //年龄
    private Date createDate; //记录的建立日期
    /**
     * 以下代码为四个字段各自的 set/get 方法， 它们提供给实体类的使用者读写本类字段的方
     * 法(这些方法可由 eclipse 自动生成)
     */
    public Long getId() { return id;}
    public void setId(Long long1) {id = long1;}
    public String getName() {return name;}
    public void setName(String string) {name = string;}
    public boolean isSex() { return sex;}
    public void setSex(boolean sex) { this.sex = sex; }
    public int getAge() {return age;}
    public void setAge(int i) {age = i;}
    public Date getCreateDate() {return createDate;}
    public void setCreateDate(Date date) {createDate = date;}
}
```

程序说明：

- ID 的类型是 Long 对象，而不是 int、long。
- createDate 字段的类型是 java.util.Date，而不是 java.sql.Date，在导入包时应注意选择。

14.2.3 数据的生成

由于数据操作是分两步走："数据库→实体类→表格显示"，实体类隔离了代码对数据库的依赖，所以"数据库→实体类"这一步就不讲了，这部分的代码与 JFace 组件的使用无关紧要，也不会影响表格组件的讲解。关于数据库方面，在后面"插件项目实战"中会有详细示例。

由于没有"数据库→实体类"这一步了，那么实体类的对象如何生成呢？因为数据记录和实体对象相对应，新创建的实体对象就相当于一个空记录，我们可以用其 set 方法一个个地将值设入实体对象中，这样就能得到带有数据的实体对象了。

为了今后便于扩展，将创建实体对象的方法集中在一个类中，这种专门负责创建对象的类又叫对象工厂。此类的代码如下：

```
//----------文件名：PeopleFactory.java ----------
/**
* 创建 PeopleEntity 对象的工厂，创建三个 PeopleEntry 对象，并装入 List 集合返回
*/
public class PeopleFactory {
    public static List getPeoples() { //工厂的静态方法
        List list = new ArrayList();
        { //第 1 个实体类对象
            PeopleEntity people = new PeopleEntity();
            //注意 id 定义时是 Long 而不是 long，仅第一个字母的大小写差别
            people.setId(new Long(1));
            people.setName("陈刚");
            people.setSex(true);
            people.setAge(28);
            people.setCreateDate(new Date()); //当前日期
            list.add(people);
        }
        { //第 2 个实体类对象
            PeopleEntity p = new PeopleEntity();
            p.setId(new Long(2));
            p.setName("周闯");
            p.setSex(false);
            p.setAge(18);
            p.setCreateDate(new Date());
            list.add(p);
        }
        { //第 3 个实体类对象
            PeopleEntity p = new PeopleEntity();
            p.setId(new Long(3));
            p.setName("陈常思");
            p.setSex(true);
            p.setAge(27);
            p.setCreateDate(new Date());
            list.add(p);
        }
        return list;
    }
}
```

程序说明：

- 在实际应用中，getPeoples 方法的代码可用从数据库中取数据的代码来替代。
- 这里的 List 不是 SWT 组件的 List，而是 Java 基础类中的 java.util.List，它是属于 Java 的集合类（Collection）。List 不是必须的，根据实际开发情况也可以用数组或其他集合类来代替，如 Set、Map 等。Java 一般都用集合 Collection 来存放数据集。
- List 是一个接口，而 ArrayList 才是实际用的类，也可以根据实际开发情况用其他

List 接口的实现类，如 Vector、LinkedList 等。一般来说，采用面向接口编程的方式，应该尽量将定义类型（如 List）比实际类型（如 ArrayList）设置得更宽泛，以便于以后的扩展。

- 在 Java 的数据库编程中集合类起着重要作用，一定要很熟悉各集合类在特性上的差别，这样才能根据实际开发情况作出适当的选择（集合类的详细资料请查阅 Java 基础书籍）。

14.2.4　数据在表格中的显示

在得到由 List 装载的包含数据信息的实体类对象后，接下来就是使用 TableViewer 来显示这些数据，实现过程一般要经过如下步骤：

- 定义一个 TableViewer 对象。在定义之时，可通过 TableViewer 构造函数提供的参数设置 TableViewer 的式样，这和 SWT 中式样的用法是一样的。
- 将 TableViewer 置于容器类中，并对其布局，这和其他 SWT 组件是一样的。
- 创建 TableViewer 中的列（TableColumn）。
- 用 setInput 方法输入数据。setInput 就像人的嘴巴，TableViewer 的数据由 setInput 方法输入。
- 设定内容器。
- 设定标签器。

这里出现了两个新概念：内容器、标签器，这是两个重要的概念，在后面的 JFace 组件中会经常用到。它们实际是 IStructuredContentProvider、ITableLabelProvider 这两个接口的实现类，它们的作用就是定义数据自 Input 进入后应该如何在 TableViewer 中显示，因此这两个类是很关键的。图 14.4 是整个数据流程的示意图。

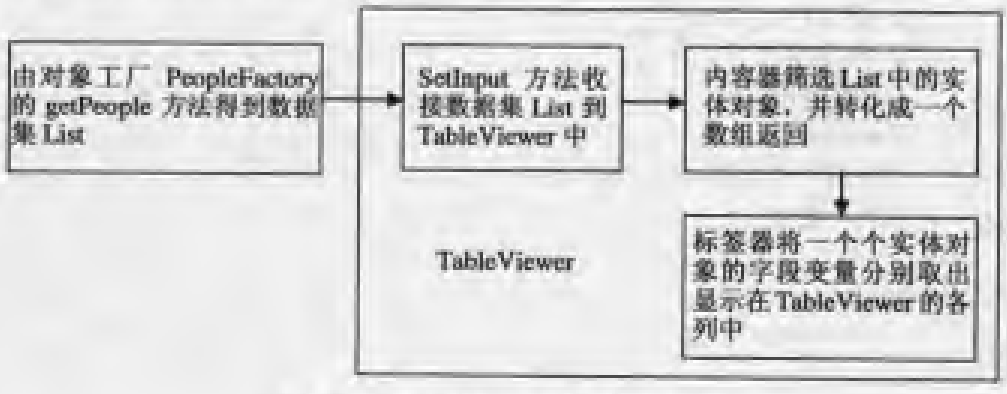


图 14.4　TableViewer 数据流示意图

一般来说，创建界面的代码都是很长的，为了阅读上的方便，可以将界面生成代码分解成一个个的自定义方法，同时将内容器和标签器也各写成一个单独的类。程序代码如下：

```
//----------文件名：TableViewer1.java----------
public class TableViewer1 {
    public static void main(String[] args) {
```

```
        TableViewer1 window = new TableViewer1();
        window.open();
    }

    private TableViewer tv; //定义一个表格对象
    public void open() {
        final Display display = new Display();
        final Shell shell = new Shell();
        shell.setSize(500, 150);
        shell.setLayout(new FillLayout());
        shell.setText("TableViewer 的实例");
        //调用自定义方法创建表格
        createTableViewer(shell);
        //第四步：设定内容器
        tv.setContentProvider(new TableViewerContentProvider());
        //第五步：设定标签器
        tv.setLabelProvider(new TableViewerLabelProvider());
        //第六步：用 setInput 输入数据
        tv.setInput(PeopleFactory.getPeoples());
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) display.sleep();
        }
    }
    /**
     * 创建表格
     */
    private void createTableViewer(Composite parent) {
        /*
         * 第一步：定义一个 TableViewer 对象。式样：MULTI 可多选、H_SCROLL 有水平
         * 滚动条、V_SCROLL 有垂直滚动条、BORDER 有边框、FULL_SELECTION 整行选择
         */
        tv = new TableViewer(parent, SWT.MULTI | SWT.BORDER
                                                | SWT.FULL_SELECTION);
        /*
         * 第二步：通过 TableViewer 中的 Table 对其布局
         */
        Table table = tv.getTable();
        table.setHeaderVisible(true); //显示表头
        table.setLinesVisible(true); //显示表格线
        TableLayout tLayout = new TableLayout(); //专用于表格的布局
        table.setLayout(tLayout);
        /*
         * 第三步：建立 TableViewer 中的列
         */
        tLayout.addColumnData(new ColumnWeightData(10));//ID 列宽 10 像素
```

```
        new TableColumn(table, SWT.NONE).setText("ID 号");
        tLayout.addColumnData(new ColumnWeightData(40));
        new TableColumn(table, SWT.NONE).setText("姓名");
        tLayout.addColumnData(new ColumnWeightData(20));
        new TableColumn(table, SWT.NONE).setText("性别");
        tLayout.addColumnData(new ColumnWeightData(20));
        new TableColumn(table, SWT.NONE).setText("年龄");
        tLayout.addColumnData(new ColumnWeightData(60));
        new TableColumn(table, SWT.NONE).setText("记录建立时间");
    }
}

//----------文件名：TableViewerContentProvidar.java----------
/**
 * 内容器。由此类对 List 集合中的数据记录进行筛选转化。此类要实现接口的三种方法，
 * 其中 getElements 是主要方法，另外两个方法很少用到，空实现就行了
 */
public class TableViewerContentProvider
                                    implements IStructuredContentProvider {
    /**
     * 此方法接收输入到表格的数据集合，然后转化成数组返回，每一个数组元素就是一个
     * 实体类对象，也即一条记录。参数 element 就是通过 setInput(getPeoples())输入的对
     * 象，因为 getPeoples()返回的是一个 List，所以参数 element 的类型也是 List
     */
    public Object[] getElements(Object element) {
        if (element instanceof List)//加一个类型判断
            return ((List) element).toArray(); //将数据集 List 转化为数组
        else
            return new Object[0]; //如非 List 类型则返回一个空数组
    }

    //此方法当 TableViewer 对象被关闭时执行，主窗口关闭了，当然 TableViewer 也跟着关闭
    public void dispose() {}
    //此方法当 TableViewer 再次调用 setInput()时执行
    public void inputChanged(Viewer v, Object oldInput, Object newInput) {}
}

//----------文件名：TableViewerLabelProvider.java----------
/**
 * 标签器。如果说内容器是对 setInput 输入的数据集作处理，那么标签器则是对数据集中实体对
 * 象中的字段数据进行处理，由标签器来决定数据记录在表格的每一列是怎么显示的
 */
public class TableViewerLabelProvider implements ITableLabelProvider {
    /**
     * 由此方法决定数据记录在表格的每一列是显示什么文字
     * @param element 实体类对象
```

```
     * @param col 列号，0 是第一列
     * @return 返回值一定要避免 null 值,否则出错
     */
    public String getColumnText(Object element, int col) {
        PeopleEntity o = (PeopleEntity) element; //要类型转换
        if (col == 0) return o.getId().toString();
        if (col == 1) return o.getName();
        if (col == 2) return o.isSex() ? "男" : "女";
        if (col == 3) return "" + o.getAge(); //加""将 int 型转为 String 型
        if (col == 4) return o.getCreateDate().toString();
        return "";
    }
    /**
     * 返回每条记录前面的图标
     */
    public Image getColumnImage(Object element, int columnIndex) {
        return null;//和 getColumnText()方法不同，此方法可返回 null 值
    }
    //----------以下方法用处不大,让它们空实现，不用管----------
    public void addListener(ILabelProviderListener listener) {}
    public void dispose() {}
    public boolean isLabelProperty(Object element, String property) {
        return false;
    }
    public void removeListener(ILabelProviderListener listener) {}
}
```

程序说明：

TableViewer 的 setInput 方法的参数类型是 Object，所以它可以接受任何类型的参数，SWT/JFace 中很多参数都是这样的，所以参数对象进入到方法内部后要进行类型转换，这是比较麻烦的一件事。因此在内容器里要将参数转换过来，如(List) element。但如果某种情况下 element 参数不是 List 类型，程序就会出错，所以最好用 element instanceof List 来作一下类型判断会比较稳妥，在 SWT/JFace 编程中很多 BUG 都出在这种地方。

还有一个出 BUG 比较多的是空值（null）的判断，实践经验表明，50%以上的 BUG 都出在空值异常上。像内容器的 getColumnImage 方法能返回 null 值，但 getColumnText 返回 null 值就会出错，所以为了程序的稳定和健壮，对方法的参数作一下空值判断是非常有必要的，除非你确定该值不可能为空值。

14.3　TableViewer 响应鼠标事件

如何让 TableViewer 的每一行响应鼠标的双击或单击事件呢？本节将解决这个问题。本节实例的效果如图 14.5 所示，双击表格中的某条记录时弹出一个提示框，框中的文字信息

显示该记录的人名。

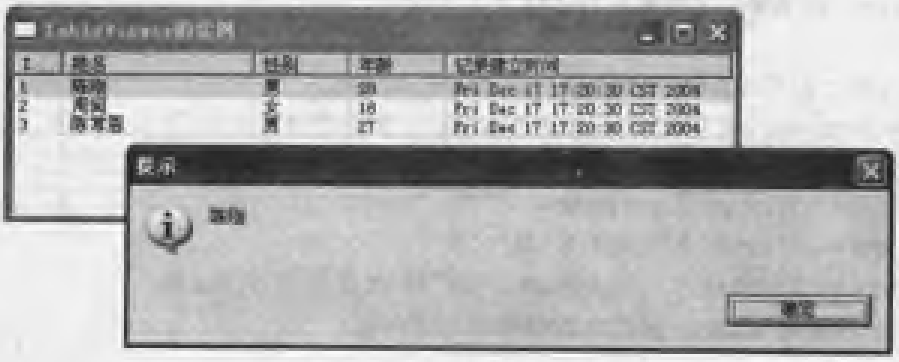


图 14.5　鼠标响应的效果图

实例将在 14.2 节的代码 TableViewer1.java 基础上进行修改（完整代码见配书光盘的 TableViewer2.java 文件），修改步骤如下。

（1）新增一个自定义方法来添加监听器。代码如下：

```
private void addListener() {
    /*
     * tv 的鼠标双击事件监听
     */
    tv.addDoubleClickListener(new IDoubleClickListener() {
        public void doubleClick(DoubleClickEvent event) {
            IStructuredSelection selection =
                                    (IStructuredSelection) event.getSelection();
            //得到记录的实体对象（要类型转换）
            PeopleEntity o = (PeopleEntity) selection.getFirstElement();
            //弹出一个提示框
            MessageDialog.openInformation(null, "提示", o.getName());
        }
    });
    /*
     * tv 的选择事件（单击）监听
     */
    tv.addSelectionChangedListener(new ISelectionChangedListener() {
        public void selectionChanged(SelectionChangedEvent event) {
        //-------事件处理代码……（省略）
        }
    });
}
```

（2）在主程序的 open()方法中调用它。代码如下：

```
tv.setInput(PeopleFactory.getPeoples());
addListener();//调用此方法
```

14.4　加上右键菜单（Action 类、ActionGroup 类、MenuManager 类）

本节我们来给表格加上如图 14.6 所示的右键菜单。

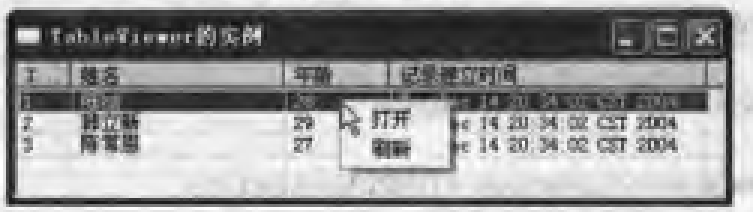


图 14.6　右键菜单的效果图

14.4.1　Action 类、ActionGroup 类、MenuManager 类介绍

SWT 中菜单是 Menu 类，本书在前面章节中已经介绍过 Menu 类的使用，在第 11 章里菜单项用 MenuItem 类来实现。但在实际项目中，同一种功能会有多种表现形式，例如 Eclipse 中的“新建”功能，它会分别出现在主菜单、主工具栏、右键菜单里。如果都是用 MenuItem 来实现，就需要写三份类似的代码，以后也要维护三份代码。当然也可以将事件处理写成外部类来共享代码，但名称、图像以及一些其他的信息写成外部类来共享则不太方便。

JFace 包中已经对以上问题提供了解决方案，JFace 提供了一个 Action 类，它将名称、图像、动作处理程序等集成在其中，这样就可以共享这些 Action 来形成菜单项、工具栏按钮等。

当然在底层最后用于 Menu 的还是 MenuItem 对象，将 Action 转化成 MenuItem 是由 MenuManager（菜单管理器）来完成的。MenuManager 简化了菜单的创建，一旦生成了 MenuManager 对象，就可以通用于菜单栏、弹出菜单、工具栏下拉菜单。

另外，Action 写成一个个的类会很零乱，JFace 又提供了一个 ActionGroup 类用于统一管理 Action，然后让外界程序通过 ActionGroup 来访问 Action。当然，ActionGroup 并非是使用 Action 所一定要的类，只是用它会更好。

14.4.2　创建 Action 和 ActionGroup

以下代码演示了如何创建 Action、ActionGroup，以及如何使用 MenuManager。

```
//--------文件名：MyActionGroup.java---------
public class MyActionGroup extends ActionGroup {
    private TableViewer tv;
    /**
     * 用来接受 TableViewer 对象的构造函数，因为在 Action 要使用到 TableViewer 对象
```

```
     * 所以一定要把 TableViewer 传进来
     */
    public MyActionGroup(TableViewer tableViewer) {
        this.tv = tableViewer;
    }
    /**
     * 生成菜单 Menu，并将两个 Action 传入
     */
    public void fillContextMenu(IMenuManager mgr) {
        /*
         * 加入两个 Action 对象到菜单管理器
         */
        MenuManager menuManager = (MenuManager) mgr;
        menuManager.add(new OpenAction());
        menuManager.add(new RefreshAction());
        /*
         * 生成 Menu 并挂在表格 table 上。menu 和 table 两个对象互为对方的参数
         */
        Table table = tv.getTable();
        Menu menu = menuManager.createContextMenu(table);
        table.setMenu(menu);
    }
    /**
     * "打开" 的 Action 类
     */
    private class OpenAction extends Action {
        public OpenAction() {
            setText("打开");
        }
        /**
         * 继承自 Action 的方法，动作代码写此方法中
         */
        public void run() {
            IStructuredSelection selection = (IStructuredSelection) tv
                                                    .getSelection();
            PeopleEntity obj = (PeopleEntity) (selection.getFirstElement());
            if (obj == null)
                MessageDialog.openInformation(null, null, "选择记录");
            else
                MessageDialog.openInformation(null, null, obj.getName());
        }
    }
    /**
     * 刷新的 Action 类
     */
    private class RefreshAction extends Action {
```

```
        public RefreshAction() {
            setText("刷新");
        }
        public void run() {
            tv.refresh();//调用表格的刷新方法
        }
    }
}
```

14.4.3　在主程序中使用 ActionGroup、MenuManager

MyActionGroup 类封装了 Action 以及 Action 和菜单 Menu 之间的交互代码。最后，我们只需要将以下代码加入到 14.2.4 小节 TableViewer1 类代码的 shell.open();语句之前即可（完整代码见配书光盘的 TableViewer3.java 及 MyActionGroup.java 文件）。

```
//生成一个 ActionGroup 对象
MyActionGroup actionGroup = new MyActionGroup(tv);
//调用 fillContextMenu 方法将按钮注入到菜单对象中
actionGroup.fillContextMenu(new MenuManager());
```

程序说明：

图 14.7 说明了在程序中是如何创建右键菜单的，在主程序生成一个 MenuManager 对象传给 ActionGroup 对象，然后再通过 ActionGroup 内部的 createContextMenu 方法生成一个菜单对象 Menu，最后用 Menu 的 add()方法将 Action 加入。

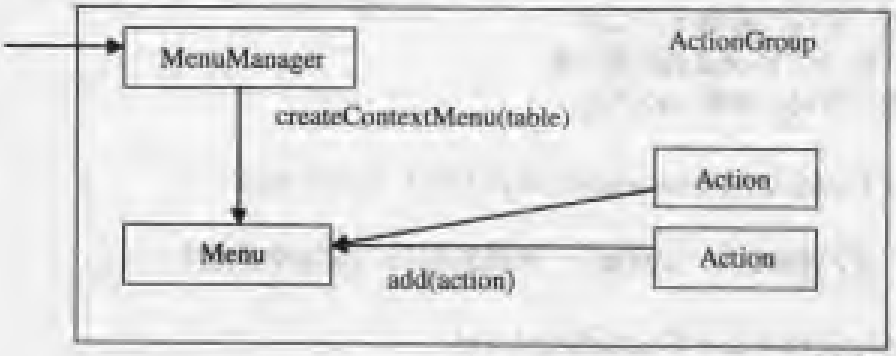


图 14.7　程序分析图

14.5　TableViewer 排序（ViewerSorter 类）

要实现表格的单击表头排序功能，一般分为如下几步（以实现 ID、姓名两列排序为例），完整代码见配书光盘的 TableViewer4.java 及 MySorter.java 文件。

14.5.1 编写排序器

TableViewer 是根据排序器 ViewerSorter 中的设置来进行排序的，所以编写 ViewerSorter 类是排序的关键。编写排序器的代码如下：

```
//------文件名：MySorter.java----------
public class MySorter extends ViewerSorter {
    //每列对应一个不同的常量，正数为升序、其相反数为降序
    public static final int ID = 1;
    public static final int NAME = 2;
    //所要排序的列
    private int sortType;
    /**
     * 构造函数，加 private 表示不能用 new MySorter()生成本类对象了
     */
    private MySorter() {}
    /**
     * 构造函数，根据不同的列来生成不同的排序对象
     */
    public MySorter(int sortType) {
        this.sortType = sortType;
    }
    /**
     * 最关键的方法：改写继承自 ViewerSorter 的方法，它的返回值是一个整数
     * 正数，则 obj1 移到 obj2 之间
     * 零，则 obj1 和 obj2 的位置不动
     * 负数，则 obj1 移到 obj2 之后
     */
    public int compare(Viewer viewer, Object obj1, Object obj2) {
        /*
         * 传入两条记录（实体类），然后依列给出它俩的先后顺序
         */
        PeopleEntity o1 = (PeopleEntity) obj1;
        PeopleEntity o2 = (PeopleEntity) obj2;
        switch (sortType) {
        case ID: {
            Long l1 = o1.getId();
            Long l2 = o2.getId();
            /*
             * compareTo 返回值有三个值 1,0,-1
             * "l1" 大于 "l2" 返回 1
             * "l1" 等于 "l2" 返回 0
             * "l1" 小于 "l2" 返回-1
             * 更详细资料请查询 JDK 帮助文档
```

```
            */
            return l1.compareTo(l2);
        }
        case -ID: {
            Long l1 = o1.getId();
            Long l2 = o2.getId();
            return l2.compareTo(l1);
        }
        case NAME: {
            String s1 = o1.getName();
            String s2 = o2.getName();
            return s1.compareTo(s2);
        }
        case -NAME: {
            String s1 = o1.getName();
            String s2 = o2.getName();
            return s2.compareTo(s1);
        }
        }
        return 0;
    }
}
```

程序说明：

排序器的代码看着挺多，其实要点就两个。

- 要用 MySorter 类生成出四个排序对象：ID 列的升序、ID 列的降序、姓名列的升序、姓名列的降序，那么 MySorter 首先就要解决如何生成这四个不同的排序对象。方法就是让不同的列对应不同的 int 值，而 int 值的相反数对应升、降序，然后根据 MySorter 类构造函数传入的 int 值就可以判断生成不同的排序器对象。
- 排序的算法由 MySorter 类的 compare 方法负责，它实际调用的是 JDK 中同类之间进行比较的 compareTo 方法，TableViewer 则根据 MySorter 返回的 int 值来进行记录的排序。总体上表示就是：JDK.compareTo → MySorter. compare → TableViewer。

14.5.2　生成排序器对象

这里要根据 MySorter 类来生成四个排序器对象：ID 列的升序、ID 列的降序、姓名列的升序、姓名列的降序，我们把它们定义成实例变量。

```
private TableViewer tv; //旧有语句
private ViewerSorter idSorter0 = new MySorter(MySorter.ID);
private ViewerSorter idSorter1 = new MySorter(-MySorter.ID);
private ViewerSorter nameSorter0 = new MySorter(MySorter.NAME);
private ViewerSorter nameSorter1 = new MySorter(-MySorter.NAME);
```

14.5.3　为表格的列添加监听器及事件处理代码

表格列是 TableColumn 对象，把原来新增 ID 列和姓名列的四句代码修改如下：

```
//          tLayout.addColumnData(new ColumnWeightData(10));
//          new TableColumn(table, SWT.NONE).setText("ID 号");
//          tLayout.addColumnData(new ColumnWeightData(40));
//          new TableColumn(table, SWT.NONE).setText("姓名");
tLayout.addColumnData(new ColumnWeightData(10));
TableColumn col1 = new TableColumn(table, SWT.NONE);
col1.setText("ID 号");
col1.addSelectionListener(new SelectionAdapter() {
    boolean sortType = true; //sortType 记录上一次的排序方式，默认为升序
    public void widgetSelected(SelectionEvent e) {
        //取反，下一次排序方式要和这一次的相反
        sortType = !sortType;
        //sortType 为真，则排序器用 idSorter0（升序）、否则用 idSorter1（降序）
        tv.setSorter(sortType ? idSorter0 : idSorter1);
    }
});

tLayout.addColumnData(new ColumnWeightData(40));
TableColumn col2 = new TableColumn(table, SWT.NONE);
col2.setText("姓名");
col2.addSelectionListener(new SelectionAdapter() {
    boolean sortType = true;
    public void widgetSelected(SelectionEvent e) {
        sortType = !sortType;
        tv.setSorter(sortType ? nameSorter0 : nameSorter1);
    }
});
```

程序说明：

程序中还考虑了这样的功能：第一次单击是升序，第二次单击则变为降序，再单击又变为升序，即每一次单击与上一次单击的排序方式相反。实现的方法是，利用一个布尔变量 sortType 来保存排序的升降，每次单击后它都将自己取反。

14.6　加上工具栏（ToolBarManager 类）

如图 14.8 所示，本节来给表格加上一个工具栏，工具栏上有六个功能按钮。完整代码见配书光盘的 TableViewer5.java 文件。

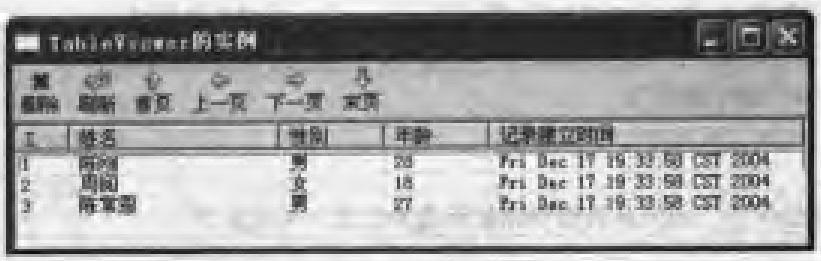


图 14.8　工具栏效果图

和 14.4 节给表格加上右键菜单的方法差不多，也是用 ActionGroup、Action，不同的是菜单用到 MenuManager，这里则用 ToolBarManager。此实例分成如下几步完成。

14.6.1　创建按钮的 Action 及将 Action 填充进工具栏的方法

将图中的按钮都写成一个个的 Action 类，关于 Action 类的写法在 14.4 节已经讲过了，只需依样扩充 MyActionGroup 类中的 Action 的个数即可。另外要给 Action 加上图像，需要使用 Action 自身的 setHoverImageDescriptor(ImageDescriptor image)方法。实现代码如下：

```
//-----文件名：MyActionGroup.java----------
public class MyActionGroup extends ActionGroup {
    private TableViewer tv;
    public MyActionGroup(TableViewer v) { this.tv = v; }
    public void fillContextMenu(IMenuManager mgr) {//将 Action 填充进右键菜单
        ……（相同代码，省略）
    }
    private final class OpenAction extends Action { // “打开”菜单项的 Action
        ……（相同代码，省略）
    }
    /**
     * 刷新的 Action 类
     */
    private final class RefreshAction extends Action {
        public RefreshAction() {
            //给 Action 设置图像，getImageDesc 为自定义方法，得到一个图像
            setHoverImageDescriptor(getImageDesc("refresh.gif"));
            setText("刷新");
        }
        public void run() {
            tv.refresh();//调用表格的刷新方法
        }
    }
    private class RemoveAction extends Action {
        public RemoveAction() {
            //正常情况下的图标
            setHoverImageDescriptor(getImageDesc("remove.gif"));
```

```
            //置灰（removeAction.setEnabled(false)）情况下的图标
            setDisabledImageDescriptor(getImageDesc("disremove.gif"));
            setText("删除");
        }
        /**
         * 这里演示了如何从表格中删除所选的记录（可选多个）
         */
        public void run() {
            //得到选择的对象集
            IStructuredSelection s=(IStructuredSelection)tv.getSelection();
            if (s.isEmpty())//判断是否有选择
                MessageDialog.openInformation(null, "提示", "请先选择");
            else
                for (Iterator it = s.iterator(); it.hasNext();) {
                    PeopleEntity o = (PeopleEntity) it.next();
                    tv.remove(o);
                    /*
                     * 这里只是从界面表格将其去除掉。
                     * 想从数据库里删除还得写相应的数据库删除操作代码
                     */
                }
        }
    }

    private class FirstAction extends Action {
        public FirstAction() {
            setHoverImageDescriptor(getImageDesc("first.gif"));
            setText("首页");
        }
        public void run() {
        /*
         * 方法分两步，
         * 第一步：从数据库中取出第一页的记录。
         * 第二步：使用 tv.setInput()将新得的数据集数据传入 TableViewer 即可。
         * 下一页、上一页、末页的实现方法与此雷同。实现方法参阅第 22 章
         */
        }
    }
    private class PrevAction extends Action {
        public PrevAction() {
            this.setHoverImageDescriptor(getImageDesc("prev.gif"));
            setText("上一页");
        }
        public void run() {}
    }
    private class NextAction extends Action {
```

```
        public NextAction() {
            this.setHoverImageDescriptor(getImageDesc("next.gif"));
            setText("下一页");
        }
        public void run() {}
    }
    private class LastAction extends Action {
        public LastAction() {
            setHoverImageDescriptor(getImageDesc("last.gif"));
            setText("末页");
        }
        public void run() {}
    }
    /**
     * 得到一个图像的方法
     */
    public ImageDescriptor getImageDesc(String fileName) {
        try {
            URL url=new URL("file:icons/" + fileName);
            return ImageDescriptor.createFromURL(url);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        return null;
    }
    /**
     * 自定义方法，生成六个 Action 对象，并通过工具栏管理器 ToolBarManager 填充进工具栏
     */
    public void fillActionToolBars(ToolBarManager actionBarManager) {
        //生成按钮，按钮就是一个个的 Action
        Action removeAction = new RemoveAction();
        Action refreshAction = new RefreshAction();
        Action firstAction = new FirstAction();
        Action preAction = new PrevAction();
        Action nextAction = new NextAction();
        Action lastAction = new LastAction();
        /*
         * 将按钮通过工具栏管理器 ToolBarManager 填充进工具栏,如果用 add(action)
         * 也是可以的，只不过只有文字没有图像，要显示图像需要将 Action 包装成
         * ActionContributionItem，在这里我们将包装的处理过程写成了一个方法
         */
        actionBarManager.add(createActionContributionItem(removeAction));
        actionBarManager.add(createActionContributionItem(refreshAction));
        actionBarManager.add(createActionContributionItem(firstAction));
        actionBarManager.add(createActionContributionItem(preAction));
        actionBarManager.add(createActionContributionItem(nextAction));
```

```
        actionBarManager.add(createActionContributionItem(lastAction));
        //更新工具栏。没有这一句，工具栏上会没有任何显示
        actionBarManager.update(true);
    }

    /**
     * 将 Action 包装成 ActionContributionItem 类的方法。实际上 Action 加入到
     * ToolBarManager 或 MenuManager 里时，也做了 ActionContributionItem 的包装。
     * 大家可以看它 ToolBarManager 的 add(IAction)的源代码即知
     */
    ActionContributionItem createActionContributionItem(IAction action) {
        ActionContributionItem aci = new ActionContributionItem(action);
        aci.setMode(ActionContributionItem.MODE_FORCE_TEXT);//显示图像+文字
        return aci;
    }
}
```

14.6.2　布局调整

在上一步创建好 ActionGroup 中的 Action 后，接下来就是要在界面里加上工具栏。我们先要将布局用 ViewForm 类来调整一下，ViewForm 也是继承自 Composite 的一个容器。原先表格是建立在 Shell 之上的，现在要在 Shell 上再插入一个 ViewForm 容器，以它为基座将工具栏和表格创建于其中，如图 14.9 所示。

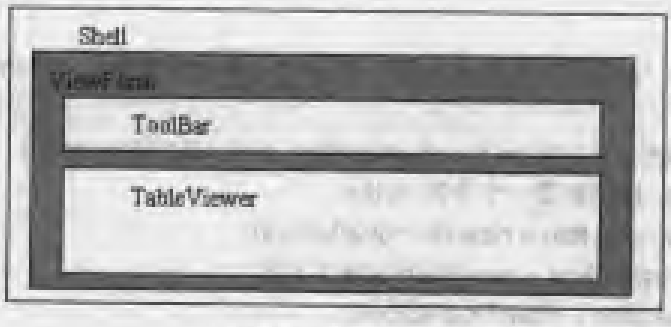


图 14.9　布局示意图

将原主程序中的 open()方法修改如下，其他代码不变：

```
shell.setLayout(new FillLayout());
/*----布局基座 ViewForm----*/
ViewForm viewForm = new ViewForm(shell, SWT.NONE);
viewForm.setLayout(new FillLayout());
//创建表格
createTableViewer(viewForm);
tv.setContentProvider(new TableViewerContentProvider());
tv.setLabelProvider(new TableViewerLabelProvider());
tv.setInput(PeopleFactory.getPeoples());
```

```
addListener();
//创建菜单栏
MyActionGroup actionGroup = new MyActionGroup(tv);
actionGroup.fillContextMenu(new MenuManager());
/*-------创建工具栏--------*/
//创建一个 ToolBar 容器
ToolBar toolBar = new ToolBar(viewForm, SWT.FLAT);
//创建一个 toolBar 的管理器
ToolBarManager toolBarManager = new ToolBarManager(toolBar);
//调用 fillActionToolBars 方法将 Action 注入 ToolBar 中
actionGroup.fillActionToolBars(toolBarManager);
/*------ 设置表格和工具栏在布局中的位置-------*/
viewForm.setContent(tv.getControl()); //主体：表格
viewForm.setTopLeft(toolBar); //顶端边缘：工具栏
```

14.7　创建一个带复选框的 TableViewer（CheckboxTableViewer 类）

带复选框的表格如图 14.10 所示，它具有如下功能：

- 单击“删除”按钮时，将所有复选框打勾的记录删除。
- 单击“刷新”按钮时，被删除记录再次复原。
- 单击“全选”按钮时，将表格中的所有记录打勾选上。
- 单击“全不选”按钮时，取消所有选择。

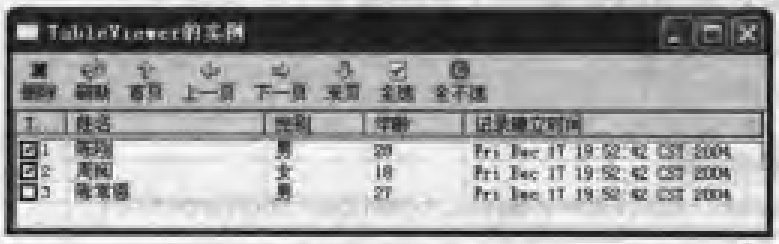


图 14.10　带复选框的表格

要完成此实例需要如下几个步骤，完整代码见配书光盘的 TableViewer6.java 文件。

14.7.1　修改界面程序将表格变为带复选框的式样

（1）在类的前部新增一个 CheckboxTableViewer 类型的实例变量。

```
private CheckboxTableViewer ctv;
```

（2）修改创建 TableViewer 的式样：多加一个 SWT.CHECK，表格变为复选框式。复选框式的表格要取得勾选的记录，还需要增加一个 CheckboxTableViewer 对象来辅助表格的

使用，因为仅 TableViewer 对象无法取得勾选记录。

```
tv = new TableViewer(parent, SWT.CHECK | SWT.MULTI | SWT.BORDER
                                                    | SWT.FULL_SELECTION);
ctv = new CheckboxTableViewer(tv.getTable());
```

（3）修改创建 MyActionGroup 的语句，将 CheckboxTableViewer 的 ctv 对象传入，因为 Action 需要用到此对象。

```
MyActionGroup actionGroup = new MyActionGroup(tv, ctv);
```

这一步完成后，因为还没有对 MyActionGroup 类作相应改动，Eclipse 会显示错误，下面开始修改 MyActionGroup 类。

14.7.2　修改 MyActionGroup 类

在原有 MyActionGroup 类的代码中作如下几处修改：

- 新增一个用于接受 CheckboxTableViewer 对象的构造函数。
- 增加“全选”和“全不选”两个 Action 类，并相应修改 fillActionToolBars 方法。
- 修改“删除 Action”，由 CheckboxTableViewer 来取得勾选的记录。因为和以前的程序共用此 Action，所以原有的处理代码还不能废弃掉，我们再加一个表格是否为复选框式样的判断，以决定使用哪种删除处理代码。

具体代码如下所示，和原代码相同部分用省略号代替：

```
public class MyActionGroup extends ActionGroup {
    private TableViewer tv;
    private CheckboxTableViewer ctv; //新增的语句
    public MyActionGroup(TableViewer v) {
        this.tv = v;
    }
    //新增一个构造函数
    public MyActionGroup(TableViewer v, CheckboxTableViewer ctv) {
        this.tv = v;
        this.ctv = ctv;
    }
    ……
    //修改原来的“删除” Action 的 run 方法
    private class RemoveAction extends Action {
        public RemoveAction() {
            ……
        }
        /**
         * 这里演示了如何从表格中删除所选的记录（可选多个）
         */
        public void run() {
```

```
            if (ctv != null) {
                Object[] checkObj = ctv.getCheckedElements(); //取得打勾的记录
                if (checkObj.length == 0) //判断是否有勾选复选框
                        MessageDialog.openInformation(null, "提示", "请先选择");
                for (int i = 0; i < checkObj.length; i++) {
                    PeopleEntity o = (PeopleEntity) checkObj[i];
                    ctv.remove(o);
                }
            } else {
                //得到选择的对象集
                IStructuredSelection s = (IStructuredSelection) tv
                                                        .getSelection();
                if (s.isEmpty())
                    MessageDialog.openInformation(null, "提示", "请先选择");
                else
                    for (Iterator it = s.iterator(); it.hasNext();) {
                        PeopleEntity o = (PeopleEntity) it.next();
                        tv.remove(o);
                    }
            }
        }
    }
    ......

    //新增的“全选”Action
    private class SelectAllAction extends Action {
        public SelectAllAction() {
            setHoverImageDescriptor(getImageDesc("selectall.gif"));
            setText("全选");
        }
        public void run() {
            if (ctv != null) ctv.setAllChecked(true); //将所有复选框打勾
        }
    }

    //新增的“全不选”Action
    private class DeselectAction extends Action {
        public DeselectAction() {
            setHoverImageDescriptor(getImageDesc("deselect.gif"));
            setText("全不选");
        }
        public void run() {
            if (ctv != null) ctv.setAllChecked(false); //取消所有复选框打勾
        }
    }
```

```
    //修改此方法将“全选”、“全不选”加入
    public void fillActionToolBars(ToolBarManager actionBarManager) {
        //生成按钮，按钮就是一个个的 Action
        ……
        Action selAllAction = new SelectAllAction();
        Action deselAction = new DeselectAction();
        //将按钮通过工具栏管理器 ToolBarManager 填充进工具栏
        ……
        actionBarManager.add(createActionContributionItem(selAllAction));
        actionBarManager.add(createActionContributionItem(deselAction));
        //更新工具栏。没有这一句，工具栏上会没有任何显示
        actionBarManager.update(true);
    }
}
```

14.8　单击修改表格单元格值（CellEditor 类、ICellModifier 接口）

前面仅仅是显示表格数据，本节谈谈如何修改数据。本节实例效果如图 14.11 所示。

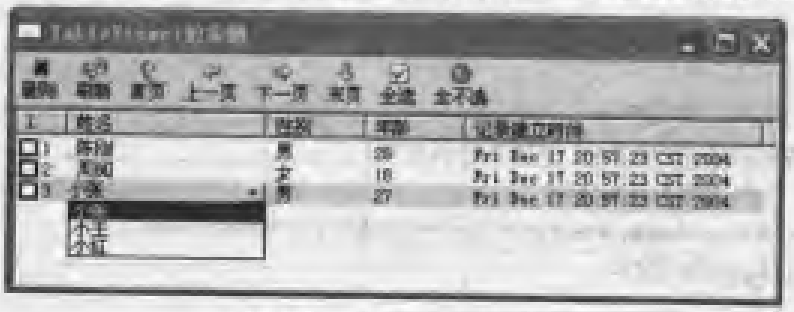


图 14.11　CellEditor 效果图

完成实例需要如下几个步骤（完整代码见配书光盘的 TableViewer7.java）。

14.8.1　创建一个字符串数组

在类前部的实例变量定义区中创建一个字符串数组，数组的元素就是修改“姓名”列时出现在下拉框中的值：

```
private String[] names = new String[] { "小张", "小王", "小红" };
```

14.8.2　修改主程序的 open()方法

在主程序的 open()方法中的“tv.setInput(PeopleFactory.getPeoples());”语句后加入如下

程序块：

```
/*
 * ------------表格修改功能的设置----------
 */
//定义每一列的别名
tv.setColumnProperties(new String[] { "0", "1", "2", "3", "4" });
//设置每一列的单元格编辑器 CellEditor
CellEditor[] cellEditor = new CellEditor[5];
cellEditor[0] = null;
cellEditor[1] = new ComboBoxCellEditor(tv.getTable(),names,SWT.READ_ONLY);
cellEditor[2] = new CheckboxCellEditor(tv.getTable());
cellEditor[3] = new TextCellEditor(tv.getTable());
cellEditor[4] = null;
tv.setCellEditors(cellEditor);
//设置单元格修改方法类 MyCellModifier
tv.setCellModifier(new MyCellModifier());
//设置第 4 列只能输入数值
Text text=(Text)cellEditor[3].getControl();
    text.addVerifyListener(new VerifyListener(){
     public void verifyText(VerifyEvent e) {
           e.doit = "0123456789".indexOf(e.text) >= 0;
     }
});
```

程序说明：

- 列的别名在 MyCellModifier 类里要用到，为了使用上的方便，用表格列的序号来做别名，当然也可以用其他的字符串。
- 和设置列别名一样，设置列的 CellEditor 编辑器也是用数组的方式，数组序号和表格列序号一一对应。
- 如果不想让某列被修改，可以将与列对应的 CellEditor 数组元素设为 null。例如，cellEditor[0] = null，则 ID 列不可被修改。
- MyCellModifier 是表格编辑器中最重要的一个类，也是最复杂的一个类，编程时一不小心就很容易出 BUG，下面将给出 MyCellModifier 类的代码。

14.8.3　创建 MyCellModifier 类

MyCellModifier 类的代码如下，将它写成一个主程序的内部类，此类也可称之为修改器。

```
private class MyCellModifier implements ICellModifier {
    /**
     * 是否可以修改此单元格，这里设置了任一单元格都可以修改。
     * @param element 表格记录对象，在这里就是 PeopleEntity。
     * @param property 列的别名
```

```
     */
    public boolean canModify(Object element, String property) {
        return true;
    }

    /**
     * 由此方法决定，当单击单元格出现 CellEditor 时应该显示什么值。
     * @param element 表格记录对象，在这里就是 PeopleEntity。
     * @param property 列的别名
     */
    public Object getValue(Object element, String property) {
        PeopleEntity o = (PeopleEntity) element;
        if (property.equals("0")) {
            return "";
        } else if (property.equals("1")) {
            return new Integer(0);
        } else if (property.equals("2")) {
            return o.isSex() ? new Boolean(true) : new Boolean(false);
        } else if (property.equals("3")) {
            return o.getAge() + "";
        } else if (property.equals("4")) { return ""; }
        return "";
    }

    /**
     * 从 CellEditor 取值得此单元格的值。
     * @param element 表格记录对象，在这里就是 PeopleEntity。
     * @param property 列的别名
     * @param value 修改的新值
     */
    public void modify(Object element, String property, Object value) {
        TableItem tableItem = (TableItem) element;
        PeopleEntity o = (PeopleEntity) tableItem.getData();
        if (property.equals("1")) {
            Integer newValue = (Integer) value;
            String name = names[newValue.intValue()];
            o.setName(name);
        } else if (property.equals("2")) {
            Boolean newValue = (Boolean) value;
            o.setSex(newValue.booleanValue());
        } else if (property.equals("3")) {
            String newValue = (String) value;
            if (newValue.equals("")) return;
            o.setAge(new Integer(newValue).intValue());
        } else {
            return;
```

```
            }
            tv.update(o, null);//或 tv.refresh();  确认并更新修改
        }
    }
```

程序说明：

- MyCellModifier 类中主要有三种方法：canModify（判断是否可编辑），getValue（判断 CellEditor 应该显示什么值），modify（将 CellEditor 修改的值更新到表格中）。
- 上述三种方法都有两个参数：Object element 和 String property，两者交叉就能确定一个单元格。
- 这里最容易出错的是 getValue 方法和 modify 方法，一定要注意 getValue 的返回值类型要和单元格的 CellEditor 具体类型相对应，否则会出错。同样 modify 方法的第三个参数 value 的实际类型，也一样会随着单元格的 CellEditor 具体类型不同而改变。它们的对照表如表 14.1 所示。

表 14.1　CellEditor的类型、getValue方法、modify方法三者的关系表

单元格编辑器 CellEditor的类型	对getValue方法返回值的数据类型的要求情况	modify方法第三个参数value的变化情况
ComboBoxCellEditor	Integer型。其值决定界面上Combo的当前显示值	Integer型
CheckboxCellEditor	Boolean型。对最初显示没有影响，但当单击修改时则会导致后面修改时的混乱，直至无法修改。例如，将上面程序都返回new Boolean(flase)，修改时将只能改为“男”，不能改为“女”	Boolean型
TextCellEditor	String型	String型

14.9　其他使用技巧

14.9.1　表格记录的过滤

建立一个继承自 ViewerFilter 的类，称之为过滤器类。下面的实例建立了一个过滤器，此过滤器的作用是在表格中只显示姓名叫“陈刚”的记录。

```
private static final class MyFilter extends ViewerFilter {
    /**
     * @param viewer 在本节就是 TableViewer
     * @param parentElement 在这时是一个包含全部记录的 Object 数组
     * @param element  当前传入，需要判断它要不要在表格显示的记录
     * @return false 值，此记录（element）不显示，True 显示
     */
    public boolean select(Viewer viewer,
                          Object parentElement,
                          Object element) {
```

```
            PeopleEntity o = (PeopleEntity) element;
            return o.getName() == "陈刚";
        }
    }
```

使用过滤器只需执行一条语句（如下所示），可以把它写入某 Action 的 run 方法中：

```
tv.addFilter(new MyFilter());  //tv 是一个 TableViewer 对象
```

14.9.2　控制表格的当前选择行

可以将以下语句写在某个事件代码中，例如写在 Action 的 run 方法中。

（1）向下移动，到底后又回到第一行。

```
Table table = tv.getTable();
int i = table.getSelectionIndex(); //取得当前所选行的序号，如没有则返回-1
table.setSelection(i + 1); //当前选择行移下一行
```

（2）向上移动，到第一行后又回到最末尾一行。

```
Table table = tv.getTable();
int i = table.getSelectionIndex();
if (i > 0) //是否超过第一行
    table.setSelection(i - 1); //向上移
else {
    int count = table.getItemCount();  //总的行数
    table.setSelection(count - 1);
}
```

14.9.3　给表格的单元格设置背景色

如下语句将使第 1 行第 2 列的单元格背景色变为红色。

```
Table table = tv.getTable(); //tv 是一个 TableViewer 对象
TableItem item = table.getItem(0); //得到第 1 行
Color color =Display.getCurrent().getSystemColor(SWT.COLOR_RED);//红色
item.setBackground(1, color); //设置此行的第 2 列为红色
table.redraw(); //重画界面
```

效果如图 14.12 所示。

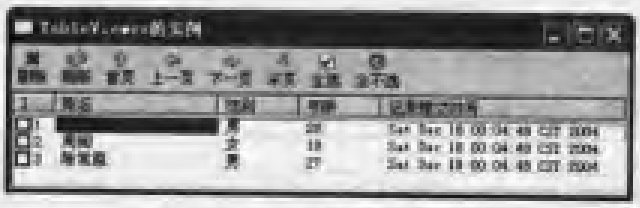


图 14.12　设置单元格背景为红色

14.9.4　加快 TableItem 和记录之间的查找速度

用以下语句可以在 TableViewer 内部为数据记录和 TableItem 之间的映射创建一个哈希表，这样可以加快 TableItem 和记录之间查找速度，这条语句必须加在 setInput 方法之前。

```
tv.setUseHashlookup(true);
```

14.10　本 章 小 结

本章是 JFace 组件的开篇之章，表格是 JFace 组件中重要而且很典型的一个组件，其中涉及了众多 JFace 的概念：内容器、标签器、过滤器、排序器、修改器，这些概念对后面 JFace 组件的学习非常重要。从本章也可以体会到 JFace 组件的非常突出的面向对象特性。

第 15 章　树（TreeViewer 类）和列表（ListViewer 类）

15.1　概　　述

前面介绍过了 SWT 中 Tree 组件的使用，和 Table 有 JFace 的扩展 TableViewer 一样，Tree 也有一个 JFace 中的扩展，就是 TreeViewer。

因为 TreeViewer 和 TableViewer 两者都是继承自同一个父类 StructuredViewer（图 15.1），所以 TreeViewer 的使用和 TableVierwer 的使用在很多地方都类似，例如，也用 setInput 方法输入数据，也有内容器、标签器以及排序器、过滤器。

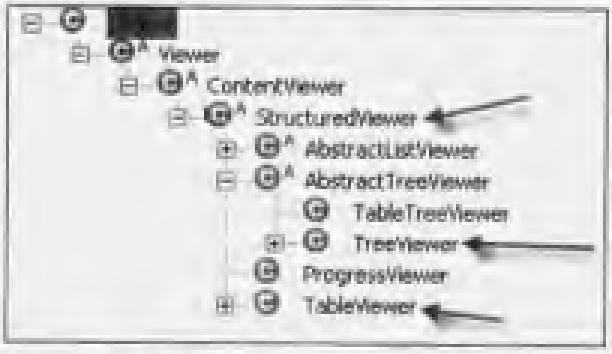


图 15.1　TreeViewer 的谱系图

15.2　前期准备：实例所用数据模型说明

15.2.1　建立国家实体和城市实体

为了编写 TreeViewer 的实例，要改造一下原来用在 TableViewer 的数据模型，在原有数据模型的基础上再增加两个实体类：国家和城市。国家包含城市，城市包含人。以下给出国家和城市的实体类代码，代码风格与原有的 PeopleEntity 类似。

```
/**
 * 国家的实体类
 */
public class CountryEntity {
    //--------三个字段----------
    private Long id; //惟一识别码，在数据库里常为自动递增的 ID 列
```

```
    private String name; //国家名
    private List cities; //此国家所包含的城市的集合，集合元素为 City 对象
     //---------两个构造函数---------
    public CountryEntity() {}
    public CountryEntity(String name) { this.name = name; }
    //---------以下为字段相应的 get/set 方法---------
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public void setChildren(List children) {this.cities = children;}
    public List getChildren() {return cities;}
}
```

```
/**
 * 城市的实体类
 */
public class CityEntity {
    //---------三个字段----------------
    private Long id; //惟一识别码，在数据库里常为自动递增的 ID 列
    private String name;//城市名
    private List peoples;//城市中的人，装在一个 List 集合中
    //---------两个构造函数--------
    public CityEntity() {}
    public CityEntity(String name) {this.name = name;}
    //---------以下为字段相应的 get/set 方法---------
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name;}
    public void setName(String name) {this.name = name;}
    public void setChildren(List children) {this.peoples = children;}
    public List getChildren() {return peoples;}
}
```

```
/**
 * 为了以后生成对象方便，我们也给 PeopleEntity 类多加一个构造函数
 */
    public PeopleEntity() {}
    public PeopleEntity(String name) { this.name = name;   }
```

15.2.2　建立树结点的接口类

树结点有两个基本特征：名称和子结点。我们将这两个特征抽象出来写成一个接口（interface），然后将要做树结点的实体类实现此接口。注意，这个接口类不是必须的，是

为了今后的操作方便，以及规范化设计才建立此接口。

```
/**
 * 树结点的接口
 */
public interface ITreeEntry {//接口名称一般以大写的 I 开头，第二个字母也大写
    //设置与得到树结点的名称
    public String getName();
    public void setName(String name);
    //设置与得到子结点集合
    public void setChildren(List children);
    public List getChildren();
}
```

15.2.3　让国家、城市、人三个实体类实现此接口

国家、城市与人的关系如图 15.2 所示。

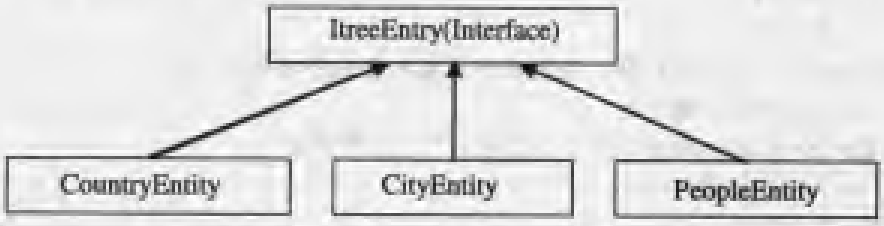


图 15.2　实体类与接口的关系图

在实体类的起始处加“implements ITreeEntry”，以表明此类实现于接口 ITreeEntry，代码如下：

```
public class CountryEntity implements ITreeEntry{
    ......
}
```

接着就要实现接口定义的 setChildren、getChildren 方法，不过原来在国家和城市的实体类已经实现了这两种方法（当初在设计这两个实体类时，笔者就考虑到了这一点）。人的实体类还没有这两种方法，但“人”在本例中是不会有子结点的，所以就让它的这两种方法空实现。

```
/**
 * 人实体，由于人不会有子结点，所以这两种方法无用，让它空实现
 */
public void setChildren(List children) {}
public List getChildren() {return null;}
```

15.3　让数据在树中显示出来

树的效果如图 15.3 所示。

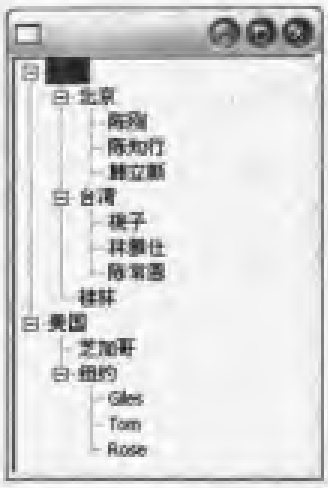


图 15.3　树的效果图

15.3.1　树的数据结构的创建

在 15.2 节，介绍了三个实体类的数据模型，接下来就要像 TableViewer 里的一样，用实体类来创建树的所有数据对象，并用集合 List 把国家、城市、人连结起来。和第 14 章一样，我们建立一个数据对象生成工厂来负责树的数据对象的创建。

```
//----------文件名：DataFactory.java----------
public class DataFactory {
    public static Object createTreeData() {
        /*
         * ----------生成数据对象----------
         */
        //生成人的数据对象
        PeopleEntity p1 = new PeopleEntity("陈刚");
        PeopleEntity p2 = new PeopleEntity("陈知行");
        PeopleEntity p3 = new PeopleEntity("韩立新");
        PeopleEntity p4 = new PeopleEntity("桃子");
        PeopleEntity p5 = new PeopleEntity("林雅仕");
        PeopleEntity p6 = new PeopleEntity("陈常恩");
        PeopleEntity p7 = new PeopleEntity("Giles");
        PeopleEntity p8 = new PeopleEntity("Tom");
        PeopleEntity p9 = new PeopleEntity("Rose");
        //生成城市的数据对象
        CityEntity city1 = new CityEntity("北京");
```

```
CityEntity city2 = new CityEntity("台湾");
CityEntity city3 = new CityEntity("桂林");
CityEntity city4 = new CityEntity("芝加哥");
CityEntity city5 = new CityEntity("纽约");
//生成国家的数据对象
CountryEntity c1 = new CountryEntity("中国");
CountryEntity c2 = new CountryEntity("美国");
/*
 * ----------------将数据对象连接起来----------------------------
 */
//--------------------人和城市的关系--------------------
{//陈刚、陈知行、韩立新是在北京的
    ArrayList list = new ArrayList();
    list.add(p1);
    list.add(p2);
    list.add(p3);
    city1.setChildren(list);
}
{ //桃子、林雅仕、陈常恩是在台湾的
    ArrayList list = new ArrayList();
    list.add(p4);
    list.add(p5);
    list.add(p6);
    city2.setChildren(list);
}
{ //Giles、Tom、Rose 是在纽约的
    ArrayList list = new ArrayList();
    list.add(p7);
    list.add(p8);
    list.add(p9);
    city5.setChildren(list);
}
//-----------------城市和国家的关系--------------------
{//北京、台湾、桂林属于中国
    ArrayList list = new ArrayList();
    list.add(city1);
    list.add(city2);
    list.add(city3);
    c1.setChildren(list);
}
{//芝加哥、纽约属于美国
    ArrayList list = new ArrayList();
    list.add(city4);
    list.add(city5);
    c2.setChildren(list);
}
```

```
        //-------将国家置于一个对象之下，这个对象可以是 List,也可以是数组-----
        {
            ArrayList list = new ArrayList();
            list.add(c1);
            list.add(c2);
            return list;
        }
    }
}
```

15.3.2　标签器和内容器的使用

TreeViewer 和 TableViewer 一样，也是用内容器和标签器来控制记录对象的显示，并且使用内容器和标签器的语句也一样。这里先给出 TreeViewer 的主程序如下：

```
public class TreeViewer1 {
    public static void main(String[] args) {
        TreeViewer1 window = new TreeViewer1();
        window.open();
    }
    public void open() {
        final Display display = new Display();
        final Shell shell = new Shell();
        shell.setSize(200, 300);
        //-----------------------------
        shell.setLayout(new FillLayout());
        Composite c = new Composite(shell, SWT.NONE);
        c.setLayout(new FillLayout());
        TreeViewer tv = new TreeViewer(c, SWT.BORDER);
        tv.setContentProvider(new TreeViewerContentProvider());
        tv.setLabelProvider(new TreeViewerLableProvider());
        //和 TableViewer 一样，数据的入口也是 setInput 方法
        Object inputObj = DataFactory.createTreeData();
        tv.setInput(inputObj);
        //-----------------------------
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
    }
}
```

15.3.3　标签器（实现 ILabelProvider 接口）

将标签器写成单独的外部类，以便于后面的实例共用，其代码如下：

```
//----------文件名：TreeViewerLableProvider.java----------
/**
 * 标签提供器。控制记录在树中显示的文字和图像等
 */
public class TreeViewerLableProvider implements ILabelProvider {
    /**
     * 记录显示的文字，不能返回 null 值
     */
    public String getText(Object element) {
        ITreeEntry entry = (ITreeEntry) element;
        return entry.getName();
    }
    /**
     * 记录显示的图像，可以返回 null 值
     */
    public Image getImage(Object element) {
        return null;
    }
    //--------以下方法暂不用，空实现---------
    public void addListener(ILabelProviderListener listener) {}
    public void dispose() {}
    public boolean isLabelProperty(Object e, String p){return false;}
    public void removeListener(ILabelProviderListener listener) {}
}
```

15.3.4　内容器（实现 ITreeContentProvider 接口）

标签器还比较简单，在 TreeViewer 中最主要、最复杂的是内容器。本例把内容器和标签器一样写成单独的外部类，以便于后面的实例共用，其代码如下：

```
//--------文件名：TreeViewerContentProvider.java----------
/**
 * 内容器。由它决定哪些对象记录应该输出在 TreeViewer 里显示
 */
public class TreeViewerContentProvider implements ITreeContentProvider {
    /**
     * 由这种方法决定树的一级显示哪些对象
     * @param inputElement  是用 tv.setInput()方法输入的那个对象
     * @return Object[]一个数组，数组中一个元素就是一个结点
     */
    public Object[] getElements(Object inputElement) {
        if (inputElement instanceof List) {
            List list = (List) inputElement;
            return list.toArray();
        } else {
```

```
            return new Object[0]; //生成一个空数组
        }
    }
    /**
     * 判断某结点是否有子结点。如果有子结点，这时结点前都有一个“+”号图标
     * @param element 需要判断是否有子结点
     * @return true 有子结点，false 无子结点
     */
    public boolean hasChildren(Object element) {
        ITreeEntry entry = (ITreeEntry) element;
        List list = entry.getChildren();
        if (list == null || list.isEmpty()) //判断是否有子结点
            return false;
        else
            return true;
    }
    /**
     * 由这种方法决定父结点应该显示哪些子结点。
     * @param parentElement 当前被单击的结点对象
     * @return 由子结点作为元素的数组
     */
    public Object[] getChildren(Object parentElement) {
        ITreeEntry entry = (ITreeEntry) parentElement;
        List list = entry.getChildren();
        if (list == null || list.isEmpty())
            return new Object[0];
        else
            return list.toArray();
    }
    //----------以下方法无用，空实现----------
    public Object getParent(Object element) {return null;}
    public void dispose() {}
    public void inputChanged(Viewer v, Object oldInput, Object newInput) {}
}
```

程序说明：

在内容器中最关键的是 getElements、hasChildren、getChildren 这三种方法。

- getElements 只在显示“一级”结点时有用。
- hasChildren 主要用于判断当前所显示的结点是否有子结点，如果有子结点则前面显示一个“+”号图标，而有“+”号的结点则可以单击展开其下一级的子结点。
- 当单击有“子”的结点时，会执行 getChildren 方法。当然展开子结点后，少不了又执行一遍子结点的 hasChildren 方法，以决定其各子结点前是否显示“+”号图标。

图 15.4 给出了内容器在启动、单击结点、关闭窗口这三种情况时的方法执行的时序图。

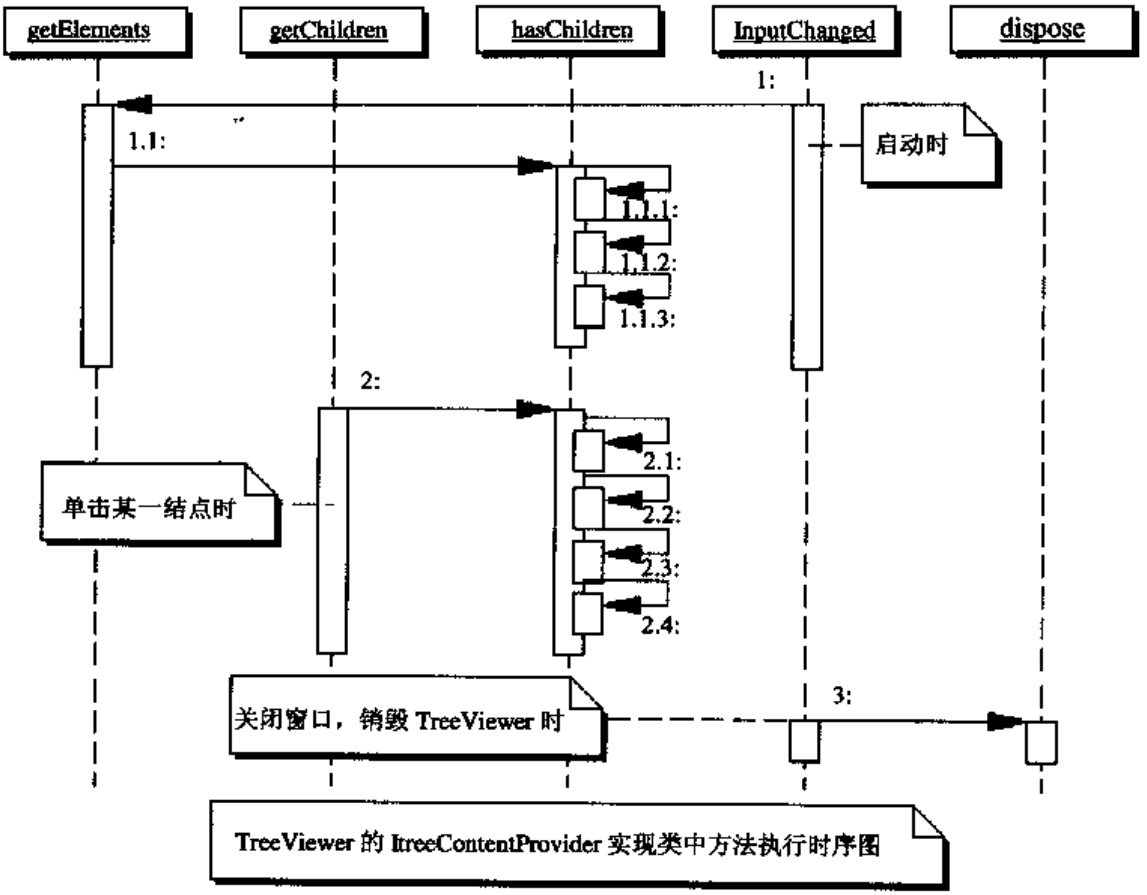


图 15.4　内容器的方法的时序图

下面以本实例来解释此图。

- 启动时：先执行一次 inputChanged 方法，此方法为空实现，跳过。再执行一次“Object[] getElements(Object inputElement)”方法，参数 inputElement 实际上就是 setInput 的入口参数的值，即 DataFactory.createTreeData()的返回值：一个包含“中国、美国”两个实体对象的 List。此 List 在 getElements 中被转化成一个数组返回，因此界面的一级结点显示的是“中国、美国”两个结点。接下来连续执行两次“boolean hasChildren(Object element)”方法，element 参数分别传入“中国、美国”两个结点对象并进行判断其是否有子结点，毫无疑问，这两个结点都有子结点，因此返回 True，界面上都显示有“＋”图标。
- 单击某一结点时：例如单击“中国”结点，会先执行一次“Object[] getChildren(Object parentElement)”方法，parentElement 参数传入的就是被单击的“中国”结点对象，然后 getChildren 方法中的程序把“中国”下的子结点取出并转化成一个数组返回，此数组包含三个结点对象“北京、台湾、桂林”。接下来连续执行三次 hasChildren 方法，传入的参数分别为“北京、台湾、桂林”这三个结点对象，并判断它们是否有子结点，如果有，则在结点前显示一个“＋”图标。
- 关闭窗口：先后会执行 inputChanged、dispose 方法。如果有需要，可以在 dispose 方法中将一些树所要用到无法自动销毁的资源 dispose 掉，例如图像 Image、字体

Font、颜色 Color 等。

15.4　给树加上右键菜单及取得结点的值

本实例的效果如图 15.5 所示。

图 15.5　右键菜单效果图

在树上加右键菜单和在 TableViewer 上加右键菜单的方法是一样的，也要用到 Action 和 ActionGroup 以及 MenuManager 类，当然程序要根据树的特点作稍微改动。

（1）实例的主程序代码如下：

```
shell.setLayout(new FillLayout());
Composite c = new Composite(shell, SWT.NONE);
c.setLayout(new FillLayout());
TreeViewer tv = new TreeViewer(c, SWT.BORDER);
tv.setContentProvider(new TreeViewerContentProvider());
tv.setLabelProvider(new TreeViewerLableProvider());
Object inputObj = DataFactory.createTreeData();
//-------------加入代码：START-----------------
//生成一个 ActionGroup 对象
MyActionGroup actionGroup = new MyActionGroup(tv);
//调用 fillContextMenu 方法将按钮注入到菜单对象中
actionGroup.fillContextMenu(new MenuManager());
//-------------加入代码：END-------------------
tv.setInput(inputObj);
```

（2）ActionGroup 类的代码如下：

```
//--------文件名：MyActionGroup.java----------
public class MyActionGroup extends ActionGroup {
    private TreeViewer tv;
    public MyActionGroup(TreeViewer treeViewer) {
        this.tv = treeViewer;
    }
    /**
```

```
    * 生成菜单 Menu，并将两个 Action 传入
    */
   public void fillContextMenu(IMenuManager mgr) {
       /*
        * 加入两个 Action 对象到菜单管理器
        */
       MenuManager menuManager = (MenuManager) mgr;
       menuManager.add(new OpenAction());
       menuManager.add(new RefreshAction());
       /*
        * 生成 Menu 并挂在树 Tree 上
        */
       Tree tree = tv.getTree();
       Menu menu = menuManager.createContextMenu(tree);
       tree.setMenu(menu);
   }
   /**
    * "打开" 的 Action 类
    */
   private class OpenAction extends Action {
       public OpenAction() {
           setText("打开");
       }
       /**
        * 继承自 Action 的方法，动作代码写此方法中
        */
       public void run() {
           IStructuredSelection selection = (IStructuredSelection) tv.getSelection();
           ITreeEntry obj = (ITreeEntry) (selection.getFirstElement());
           if (obj != null)
               MessageDialog.openInformation(null, null, obj.getName());
       }
   }
   /**
    * 刷新的 Action 类
    */
   private class RefreshAction extends Action {
       public RefreshAction() {
           setText("刷新");
       }
       public void run() {
           tv.refresh();//调用 TreeViewer 的刷新方法
       }
   }
}
```

15.5　树结点的展开、收缩、新增、删除、修改

展开结点用 expandToLevel 或 expandAll 方法；收缩结点用 collapseToLevel 或 collapseAll 方法；新增结点用 add 方法；删除结点用 remove 方法；修改结点只需要取出结点对象，作相应修改后，用 refresh 方法将结点刷新一下即可。

本实例的效果如图 15.6 所示。

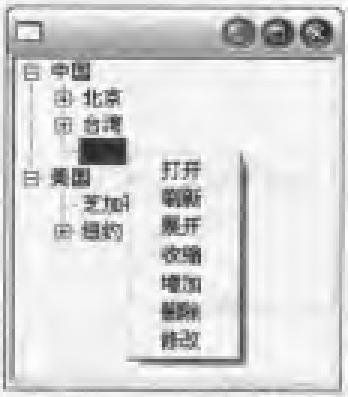


图 15.6　实例效果图

这几个功能的增加只需要修改 MyActionGroup 类，增加相应的 Action 即可，代码如下：

```
//-------文件名：MyActionGroup.java---------
public class MyActionGroup extends ActionGroup {
    private TreeViewer tv;
    public MyActionGroup(TreeViewer treeViewer) {
        this.tv = treeViewer;
    }
    /**
     * 生成菜单 Menu，并将两个 Action 传入
     */
    public void fillContextMenu(IMenuManager mgr) {
        /*
         * 加入两个 Action 对象到菜单管理器
         */
        MenuManager menuManager = (MenuManager) mgr; //类型转换
        menuManager.add(new OpenAction());
        menuManager.add(new RefreshAction());
        menuManager.add(new ExpandAction());
        menuManager.add(new CollapseAction());
        menuManager.add(new AddEntryAction());
        menuManager.add(new RemoveEntryAction());
        menuManager.add(new ModifyEntryAction());
        /*
```

```
        * 生成 Menu 并挂在树 Tree 上
        */
        Tree tree = tv.getTree();
        Menu menu = menuManager.createContextMenu(tree);
        tree.setMenu(menu);
    }
    /**
     * "打开" 的 Action 类
     */
    private class OpenAction extends Action {
        public OpenAction() {
            setText("打开");
        }
        /**
         * 继承自 Action 的方法，动作代码写此方法中
         */
        public void run() {
            ITreeEntry obj = getSelTreeEntry();
            if (obj != null)
                MessageDialog.openInformation(null, null, obj.getName());
        }
    }
    /**
     * 刷新的 Action 类
     */
    private class RefreshAction extends Action {
        public RefreshAction() {
            setText("刷新");
        }
        public void run() {
            tv.refresh();//调用 TreeViewer 的刷新方法
        }
    }
    /**
     * 展开当前结点的 Action 类
     */
    private class ExpandAction extends Action {
        public ExpandAction() {
            setText("展开");
        }
        public void run() {
            ITreeEntry obj = getSelTreeEntry();
            if (obj != null)
                tv.expandToLevel(obj, 1); //只展开一层，设值可以超过实际树的层数
        }
    }
```

```
    /**
     * 收缩当前结点的 Action 类
     */
    private class CollapseAction extends Action {
        public CollapseAction() {
            setText("收缩");
        }
        public void run() {
            ITreeEntry obj = getSelTreeEntry();
            if (obj != null)
                tv.collapseToLevel(obj, -1); //-1 为将当前结点的所有子结点收缩
        }
    }
    /**
     * 给当前结点增加一个子结点的 Action 类
     */
    private class AddEntryAction extends Action {
        public AddEntryAction() {
            setText("增加");
        }
        public void run() {
            ITreeEntry obj = getSelTreeEntry();
            if (obj == null || obj instanceof PeopleEntity)
                return;
            InputDialog dialog = new InputDialog(null, "给当前结点增加一个子结点", "输入名称",
"abc", null);
            if (dialog.open() == InputDialog.OK) {//如果单击 OK 按钮
                String entryName = dialog.getValue(); //得到 Dialog 输入值
                /* 根据单击结点的不同类型生成相应的子结点*/
                ITreeEntry newEntry = null;
                if (obj instanceof CountryEntity)
                    newEntry = new CityEntity(entryName);
                else if (obj instanceof CityEntity)
                    newEntry = new PeopleEntity(entryName);
                /*在增加子结点之前将父结点展开*/
                if (!tv.getExpandedState(obj))
                    tv.expandToLevel(obj, 1);
                tv.add(obj, newEntry);//增加结点
            }
        }
    }
    /**
     * 删除结点的 Action 类
     */
    private class RemoveEntryAction extends Action {
        public RemoveEntryAction() {
```

```
            setText("删除");
        }
        public void run() {
            ITreeEntry obj = getSelTreeEntry();
            if (obj == null)
                return;
            tv.remove(obj);
        }
    }
    /**
     * 修改结点名称的 Action 类
     */
    private class ModifyEntryAction extends Action {
        public ModifyEntryAction() {
            setText("修改");
        }
        public void run() {
            ITreeEntry obj = getSelTreeEntry();
            if (obj == null)
                return;
            InputDialog dialog = new InputDialog(null, "修改结点", "输入新名称", obj.getName(),
null);
            if (dialog.open() == InputDialog.OK) {
                String entryName = dialog.getValue();
                obj.setName(entryName);
                tv.refresh(obj); //刷新结点
            }
        }
    }
    /**
     * 自定义方法：取得当前选择的结点
     */
    private ITreeEntry getSelTreeEntry() {
        IStructuredSelection selection = (IStructuredSelection) tv.getSelection();
        ITreeEntry entry = (ITreeEntry) (selection.getFirstElement());
        return entry;
    }
}
```

程序说明：

- 为了复用代码，将取得当前选择结点的几条语句写成了一种方法。
- 在对结点对象操作时，要记得进行空值判断，因为树的大部分方法的参数都不支持空值，会报异常并且中断程序。

15.6　ListViewer 类

15.6.1　ListViewer 简介

在前面介绍过 TableViewer 和 TreeViewer 之后，ListViewer 就没有什么新鲜的知识了。如果把 TableViewer 设置成一列，就成了 ListViewer 的样子。当然 ListViewer 类并非由 TableViewer 类简化而来，它是基于 SWT 的 List 组件的一个扩展，其谱系图如图 15.7 所示。

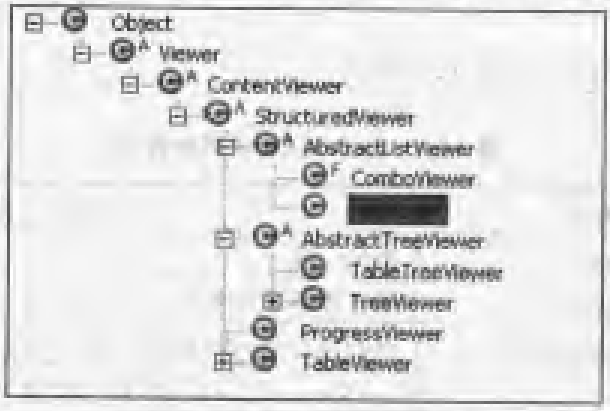


图 15.7　ListViewer 的谱系图

15.6.2　ListViewer 的实例

ListViewer 和 TableViewer、TreeViewer 一样也有自己的内容器和标签器。其内容器和 TableViewer 是一样的，甚至可以不用改动地挪用 TableViewer 的内容器，以及 TreeViewer 的标签器。在本实例输入 setInput 方法的数据是用第 15 章的数据，但只借用了两个国家对象。实例的具体代码如下：

```
shell.setLayout(new FillLayout());
ListViewer lv = new ListViewer(shell, SWT.BORDER);
lv.setContentProvider(new TableViewerContentProvider());
lv.setLabelProvider(new TreeViewerLableProvider());
lv.setInput(DataFactory.createTreeData());
```

实例运行的效果如图 15.8 所示。

可以用 add 方法往列表中加入新项。例如，在 setInput 方法后新增如下两句：

```
lv.add(new CountryEntity("英国"));
lv.add(new CountryEntity("法国"));
```

则运行效果如图 15.9 所示。

图 15.8　ListViewer 的效果图

图 15.9　增加两项的 ListViewer 效果图

15.6.3　ListViewer 常用方法

ListViewer 常用的方法，如表 15.1 所示。

表 15.1　ListViewer常用的方法

ListViewer常用方法	功　能
void add(Object element)	添加一项
void add(Object[] elements)	通过数组加入多项
Object getElementAt(int index)	返回序号为index的项（是一个对象）
List getList()	ListViewer是List的扩展，此方法可得到内含的List对象
void remove(Object element)	从ListViewer移去一项
void remove(Object[] elements)	从ListViewer移去数组中的几项

15.7　本 章 小 结

TreeViewer 的使用方式与 TableViewer 有很多地方都是非常相似的，TreeViewer 也有自己的过滤器和排序器，具体使用方式请参照 TableViewer。TreeViewer 中比较复杂的内容器，熟悉内容器是掌握 TreeViewer 的要点。

ListViewer 也有过滤器和排序器，它们的使用方法和 TableViewer 是一样的，本章没有再详细讲述。笔者在实际开发中很少用到 ListViewer，简单的开发需求，用 SWT 的 List 就够了。对于复杂的开发需求，把 TableViewer 设置成一行也能满足，而且还便于扩展成多列。如果开发需求很明确：就只要一个列表，而且数据操作比较复杂、数据量比较多，这时就比较适合用 ListViewer。

第16章 对 话 框

16.1 对话框（Dialog 类）

16.1.1 对话框简介

对话框是软件系统最常用到的界面，可以说无处不在。从界面结构来说，对话框主要由主体的界面组件和底部窗体按钮组成，图 16.1 就是一个标准的对话框。

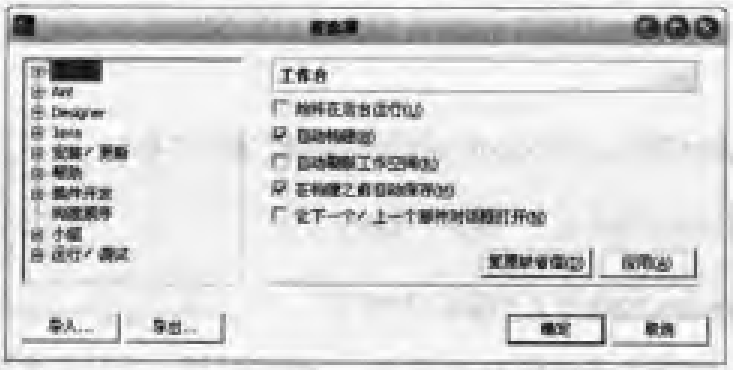


图 16.1 对话框示例

在这之前的实例中，已经频繁地用到了 MessageDialog.openInformation 方法来弹出一个简单的对话框。MessageDialog 只是 Dialog 中的一种，在 Dialog 中还有很多其他的对话框可以使用，如图 16.2 所示。本节将介绍如何通过继承 Dialog 类来实现一个自定义对话框。

图 16.2 Dialog 的谱系图

16.1.2　信息提示框（MessageDialog 类）

在这之前 MessageDialog 类已经经常被用到，本小节给出它更全面一些的使用方法。

1．void openInformation(Shell parent, String title,String message)

例子：MessageDialog.openInformation(shell,"标题","提示信息")，它的三个参数都可以接受 null 值。此方法仅用于提示信息，如图 16.3 所示。

图 16.3　openInformation 的效果图

2．boolean openConfirm(Shell parent, String title, String message)

例子：

```
boolean b = MessageDialog.openConfirm(shell, "标题", "提示信息");
if (b)
    System.out.println("你单击了 "确定" 按钮");
else
    System.out.println("你单击了 "取消" 按钮");
```

如图 16.4 所示，其界面上比图 16.3 多了一个取消按钮，并且左边变成了问号图标。

图 16.4　openConfirm 的效果图

3．boolean openQuestion (Shell parent, String title, String message)

使用方法和 openConfirm 一模一样，只是界面按钮变成了“是”、“否”，如图 16.5 所示。

图 16.5　openQuestion 的效果图

4．void openError(Shell parent, String title, String message)

例子：MessageDialog.openError(shell, "标题", "提示信息");

错误提示框，界面上换成了明显的红叉图标，如图 16.6 所示。

图 16.6　openError 的效果图

5. void openWarning (Shell parent, String title, String message)

例子：MessageDialog. openWarning (shell, "标题", "提示信息");

警告提示框，界面上换成了明显的警告图标，如图 16.7 所示。

图 16.7　openWarning 的效果图

16.1.3　输入值对话框（InputDialog 类）

格式：

```
InputDialog(Shell parentShell,
            String dialogTitle,
            String dialogMessage,
            String initialValue,
            IInputValidator validator)
```

参数说明：parentShell 是一个 Shell，可以接受 null 值；dialogTitle 为对话框的标题；dialogMessage 为对话框中的提示文字；initialValue 设置初始值；validator 是一个对输入值的验证类。

例子：InputDialog 和 MessageDialog 的使用不同，前者没有提供静态方法来打开窗口，必须要先创建一个 InputDialog 对象，然后再使用 open 方法弹出窗口。open 方法执行后，程序会挂起，直到退出 InputDialog 对话框，最后 open 方法的返回值（字符串）就是 InputDialog 中的输入值。实例代码如下：

```
//——文件名：InputDialog1.java——————
InputDialog dialog = new InputDialog(shell, "标题", "请输入值", "1", null);
if (dialog.open() == InputDialog.OK) {
    String vlaueStr = dialog.getValue();
    System.out.println("输入值：" + vlaueStr);
}
```

它的运行效果如图 16.8 所示。

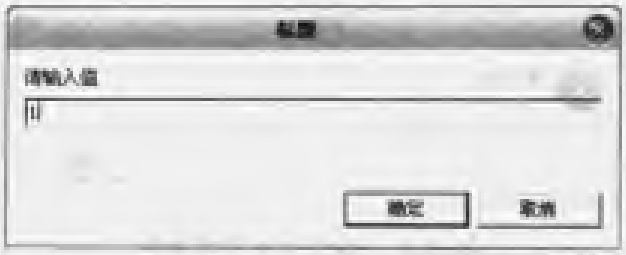


图 16.8　InputDialog 的效果图

在上例中，最后一个参数 validator 是设成了 null，也就是说不使用验证类对输入的值进行有效性验证。现在我们加入一个验证类来验证输入值的有效性：本例规定只能输入 0~100 之间的数值。其效果如图 16.9 所示，当输入值无效时，窗口下方会出现文字提示，并且“确定”按钮置灰。

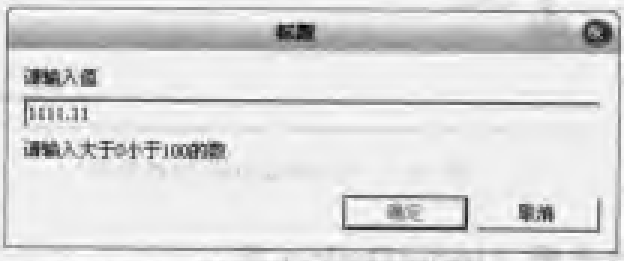


图 16.9　加了验证类的 InputDialog 的效果图

实例代码如下：

```
/**
 * 值的验证类
 */
class MyValidator implements IInputValidator {
    /**
     * 返回 null 值表示值(newText)有效通过
     * 返回其他字符串（包括""这样的空字符)表示值无效
     */
    public String isValid(String newText) {
        float value = 0;
        try {
            value = Float.valueOf(newText).floatValue();
        } catch (java.lang.NumberFormatException e) {
            return "请输入数值";
        }
        if (value > 0 && value < 100)
            return null;//返回 null 表示 newText 合法
        else
            return "请输入大于 0 小于 100 的数";
    }
}
```

然后还要将 InputDialog 对象的生成语句修改一下，将验证类传入：

```
InputDialog dialog = new InputDialog(shell, "标题", "请输入值", "1", new MyValidator());
```

16.1.4 自定义对话框（Dialog 类）

仅仅是使用 MessageDialog、InputDialog 等 JFace 中现成的对话框类是无法满足实际项目开发需要的，很多时候都需要自己定制对话框。自定义对话框只要在 Dialog 类的基础上作扩展就行了。

1. 起步：扩展 Dialog 类

先继承 Dialog 类来建立一个最简单的 Dialog 子类，代码如下：

```
//——文件名：MyDialog.java————
/**
 * 注意：此继承的 Dialog 属于 JFace 包，而 SWT 包也有一个 Dialog 类，
 * 因此在 import 时，一定要注意选择区分
 */
public class MyDialog extends Dialog {
    /**
     * 构造函数
     */
    public MyDialog(Shell parentShell) {
        super(parentShell);
    }
    /**
     * 在这个方法里构建 Dialog 中的界面内容
     */
    protected Control createDialogArea(Composite parent) {
        return null;
    }
}
```

这样就建立了一个自定义对话框：MyDialog，它的使用方法和 JFace 自带的 InputDialog 一样，如下：

```
MyDialog dialog = new MyDialog(shell);
dialog.open();
```

运行效果如图 16.10 所示。

2. 创建界面组件

在上图可以看到界面中除了两个 Dialog 类自带的按钮，还没有任何界面组件。要加入界面组件，需要在 MyDialog 类的 createDialogArea 方法中写入相应的代码。这些代码的写法和以前的界面创建方法是一样的，只不过以前是用 Shell 作为容器创建组件，现在是在 createDialogArea 方法的参数“Composite parent”上创建组件。

本实例将建立一个较简单的对话框界面：加入一个文字标签和一个文本框，如图 16.11 所示。

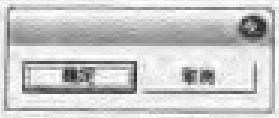


图 16.10　实例效果图

图 16.11　加入文本框后的效果图

完成此实例只要修改 createDialogArea 方法，将界面组件的创建代码加入，代码如下：

```
protected Control createDialogArea(Composite parent) {
    /*
     * 不要直接在 parent 上创建组件，否则容易导致界面布局混乱。
     * 应该向本例再插入一个面板 topComp，并在此面板创建及布局
     */
    Composite topComp = new Composite(parent, SWT.NONE);
    //应用 RowLayout 面局
    topComp.setLayout(new RowLayout());
    //加入一个文本标签
    new Label(topComp, SWT.NONE).setText("请输入：");
    //加入一个文本框
    Text text = new Text(topComp, SWT.BORDER);
    //用 RowData 来设置文本框的长度
    text.setLayoutData(new RowData(100, -1));
    return topComp;
}
```

createDialogArea 方法的返回值类型要求是 Control，其谱系图如图 16.12 所示。由图可见 Composite 是 Control 的子类，所以上面的程序中返回 topComp 对象也是可以的。

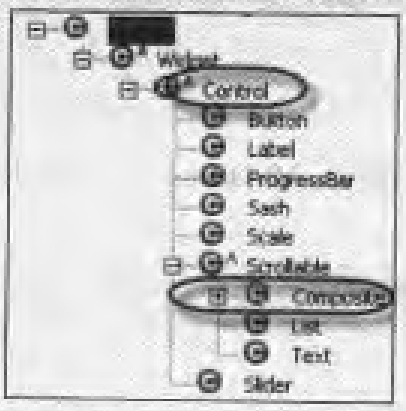


图 16.12　Control 的谱系图

3. 改变对话框的式样

在图 16.11 中，可看到窗口的右上角只有一个关闭按钮，如果想让它有最小化、最大化按钮，并且窗口可用鼠标改变大小，则只需要在 MyDialog 类中新增一个 getShellStyle 方法即可。getShellStyle 是父类 Dialog 中的方法。

```
/**
 * 改写这个父类 Dialog 的方法可以改变窗口的默认式样
 */
protected int getShellStyle() {
        /*
         * super.getShellStyle()得到原有的式样
         * SWT.RESIZE：窗口可以变大小
         * SWT.MAX：窗口可最大化、最小化
         */
        return super.getShellStyle() | SWT.RESIZE | SWT.MAX;
}
```

运行效果如图 16.13 所示。如果让此方法返回 SWT.NONE，如下：

```
protected int getShellStyle() {
        return SWT.NONE;
}
```

则得到如图 16.14 所示的效果图。

图 16.13 加最大化、最小化按钮后的效果图

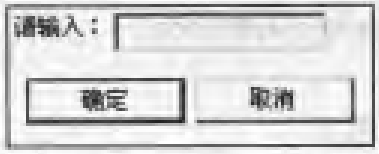


图 16.14 SWT.NONE 式样的效果图

4. 定制对话框的按钮

MyDialog 中默认显示“确定”、“取消”两个按钮，我们也可以改变按钮显示的文字和增加删除按钮，这需要在 MyDialog 中改写父类的两种方法：createButton 和 initializeBounds。

（1）实例 1

如图 16.15 所示，本例将在窗口中增加一个名为“应用”的按钮，并将原有的“确定”、“取消”按钮文字改为 OK、Cancel。

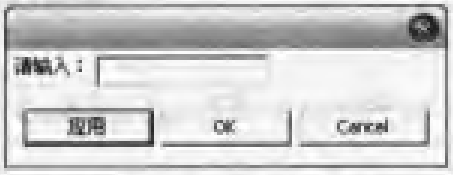


图 16.15 自定义对话框的按钮

实例代码如下：

```
/**
 * 改写父类创建按钮的方法，使其失效
 */
protected Button createButton(Composite parent, int buttonId,
```

```
                                    String buttonText, boolean defaultButton) {
        return null;
}
/**
 * 改写父类的 initializeBounds 方法，并利用父类的 createButton 方法建立按钮
 */
public static final int APPLY_ID = 101; //自定义“应用”按钮的 ID 值(整型常量)
protected void initializeBounds() {
        /*
         * createButton(Composite parent, int id, String label,boolean defaultButton)
         * 参数 parent：取得放置按钮的面板
         * 参数 ID：定义按钮的 ID 值
         * 参数 label：按钮上的文字
         * 参数 defaultButton：是否为 Dialog 的默认按钮
         */
        super.createButton((Composite) getButtonBar(), APPLY_ID, "应用", true);
        super.createButton((Composite) getButtonBar(), IDialogConstants.OK_ID, "确定", false);
        super.createButton((Composite)  getButtonBar(),  IDialogConstants.CANCEL_ID,  "取 消 ",
false);
        super.initializeBounds();
}
```

（2）实例 2

如果只是想增加一个按钮，而不想对 Dialog 原有的“确定”、“取消”按钮作任何改变，则还有一种更简单的方法：不必像上面那样改写父类的两种方法，而只需要改写 initializeBounds 这一个父类方法。

```
protected void initializeBounds() {
        super.createButton((Composite) this.getButtonBar(),APPLY_ID, "应用", true);
        super.initializeBounds(); //此两个语句顺序不能调换，否则界面将没有任何变化
}
```

其运行效果如图 16.16 所示。

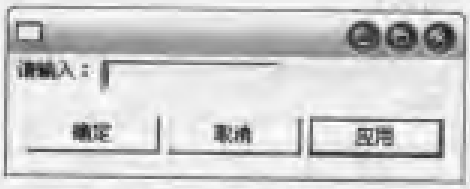


图 16.16　实例二的效果图

5. 改变自定义对话框的大小

在类中继承并改写父类的 getInitialSize 即可，代码如下：

```
protected Point getInitialSize() {
        Point p = super.getInitialSize();
        p.x = 300; //宽度（单位：像素）
```

```
    p.y = 400; //长度（单位：像素）
    return p;
}
```

16.1.5 对话框的设置与取值

在实际开发中，经常需要设置对话框中组件的初始值，并在单击对话框的“确定”按钮后，取出对话框中的值，然后执行相应的处理程序。为了不影响以前的代码，本例先将MyDialog.java 复制一份起名 MyDialog2.java，然后在 MyDialog2.java 的基础上修改，修改的方法如下：

（1）先在 MyDialog2 类中定义一个 String 类型的实例变量用于保存文本框的值，并将文本框对象也写成实例变量。

```
private String textValue; //类的实例变量
private Text text;//将文本写为类实例变量，否则其他方法无法访问它
```

（2）在 MyDialog2 类中增加 textValue 的 set/get 方法。

```
public String getTextValue() {return this.textValue;}
public void setTextValue(String string) {this.textValue = string;}
```

（3）在 MyDialog2 类的 createDialogArea 方法中，先将 Text 对象的定义语句改写如下：将 textValue 变量的值设成 Text 的初值，这时要注意对 textValue 作空值判断，给文本框设置空值是会出错的。

```
text = new Text(topComp, SWT.BORDER);
text.setText(textValue==null?"":textValue);
```

（4）在 MyDialog2 类中改写父类的 buttonPressed 方法，单击对话框底部按钮会执行这种方法，方法的参数 buttonId（int 型）是被单击按钮的 ID 值。

```
protected void buttonPressed(int buttonId) {
    //如果是点了 OK 按钮，则将值取回变量
    if (buttonId == IDialogConstants.OK_ID)
        textValue = text.getText();
    super.buttonPressed(buttonId);
}
```

（5）经过以上四步就完成了 MyDialog2 中设值、取值的工作。接着来看如何使用 MyDialog2，代码如下：

```
MyDialog2 dialog = new MyDialog2(shell);
dialog.setTextValue("你好，中国"); //设值
if (dialog.open() == IDialogConstants.OK_ID) {//是否单击“确定”按钮
    String str = dialog.getTextValue(); //取值
    System.out.println(str);
}
```

也许读者会有这样的想法：为什么要在 MyDialog2 类中加一个 String 类型的 textValue 变量呢？难道直接对文本框对象 text 进行操作不行吗？例如，省去 textValue 变量和它的 set/get 方法，而将 text 对象设成 public 公布出来，然后这样使用：

```
MyDialog2 dialog = new MyDialog2(shell);
dialog.text.setText("你好，中国"); //设值
if (dialog.open() == IDialogConstants.OK_ID) {//单击“确定”按钮
    String str = dialog.text.getText (); //取值
    System.out.println(str);
}
```

上面的代码好像简洁了不少，可惜这是行不通的。因为在执行 dialog.open()方法时，对话框界面上的组件才开始创建，也就是说在 open 方法之前还不能使用 text 文本框对象，因此 dialog.text.setText("你好，中国");这一句会出错。当单击“确定”按钮 dialog.open()方法执行完毕，对话框退出，这时对话框中的组件将自动销毁，所以在 dialog.open()之后也不能使用 text 文本框对象，也就是说 String str = dialog.text.getText ();也会出错。

正是由于以上原因，本例才不得不建立实例变量 textValue 来保存 text 文本框的值。

16.1.6　带提示栏的对话框（TitleAreaDialog 类）

1. 创建

只要将继承的父类由 Dialog 改为 TitleAreaDialog 就行了。代码如下：

```
//------文件名：MyTitleAreaDialog.java----------
public class MyTitleAreaDialog extends TitleAreaDialog {
    ......
}
```

2. 使用方法

其他代码完全不变，也可以运行。然而 TitleAreaDialog 的独特之处就是，在对话框的顶部多了一片可以显示文字图片的区域，不用岂不可惜？使用的方法很简单，仅多写几个语句，如下所示：

（1）实例 1，效果如图 16.17 所示。

```
setTitle("对话框标题");
setMessage("请在文本框中输入文字");
```

（2）实例 2，效果如图 16.18 所示。

```
setTitle("对话框标题");
setErrorMessage("输入错误");
```

（3）实例 3，效果如图 16.19 所示。

```
setTitle("对话框标题");
setMessage("对话框的说明文字", IMessageProvider.INFORMATION);
```

图 16.17 实例 1 的效果图

图 16.18 实例 2 的效果图

IMessageProvider 中的式样除了 INFORMATION，还有 NONE、WARNING、ERROR 三种，其中 NONE 式样的效果和“setMessage(String str);”相同，ERROR 式样的效果和“setErrorMessage (String str);”相同。

图 16.19 实例 3 的效果图

16.2 向导式对话框（WizardDialog 类）

16.2.1 向导式对话框简介

向导式对话框是一种非常友好的界面，它能引导用户一步步地输入信息，Eclipse 的“新建项目”就是这样的向导式对话框。在 Eclipse 中向导式对话框的开发是很简单的，它由 WizardDialog 类来创建向导式对话框，WizardDialog 谱系图如图 16.20 所示。

图 16.20 WizardDialog 谱系图

当然仅仅 WizardDialog 还是不够的，还要有 Wizard 类、WizardPage 类来辅助使用。向导式对话框中每一步显示的页面由 WizardPage 类负责创建，Wizard 类则负责管理 WizardPage 页面。WizardDialog 最简单，它和 InputDialog 等的使用相同，所以在向导式对话框的创建中 Wizard 和 WizardPage 的编程才是重点。WizardDialog、Wizard、WizardPage 三者关系如图 16.21 所示。

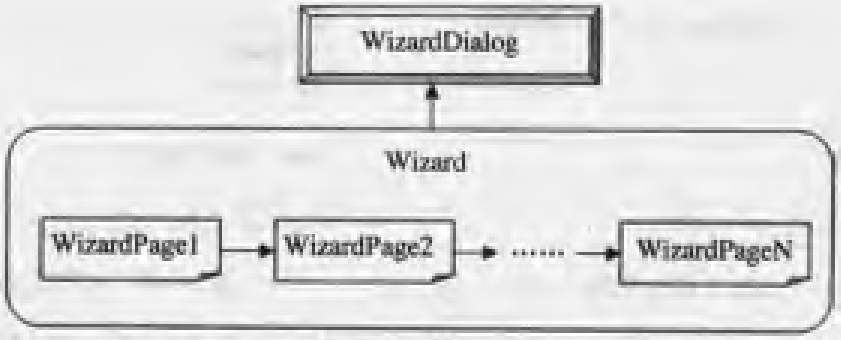


图 16.21　WizardDialog、Wizard、WizardPage 的关系图

16.2.2　向导式对话框实例

本小节将实现一个简单的向导式对话框，此对话框用于收集用户的资料。它有两个页面，第一个页面（图 16.22）收集姓名和 Email，第二页面（见图 16.23）收集学校，并且当单击“完成”按钮结束向导后，输出这些收集到的信息（见图 16.24）。

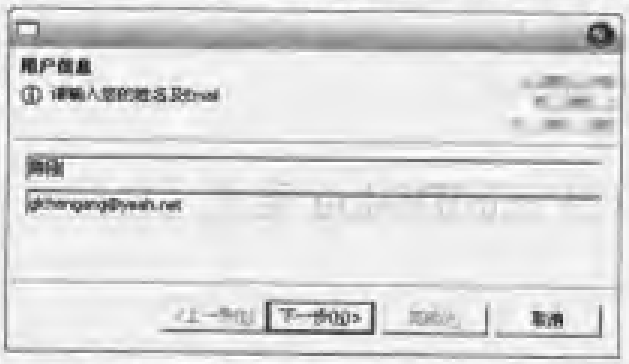


图 16.22　向导式对话框实例（第一个页面）

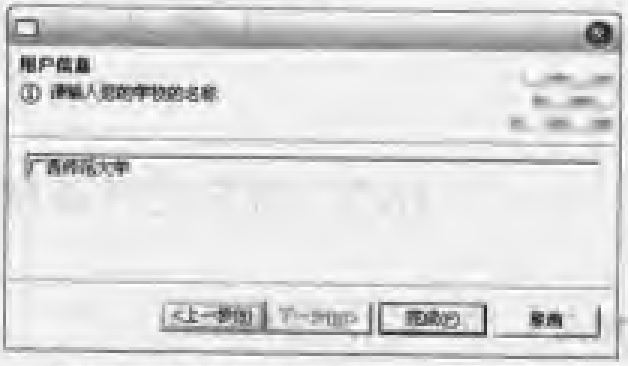


图 16.23　向导式对话框实例（第二个页面）

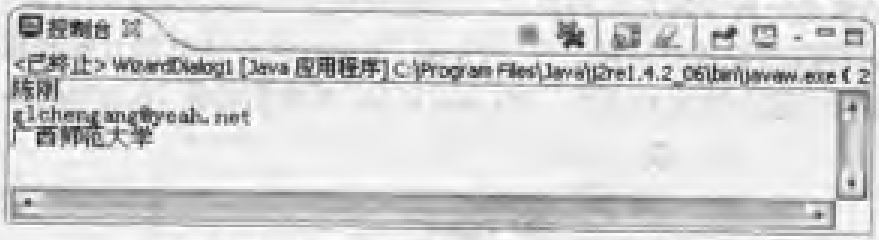


图 16.24　单击“完成”按钮后控制台的输出

从这个实例可以了解到：如何创建向导式对话框、如何给向导式对话框设初值、如何对值做验证、如何取得向导式对话框中的值。实际开发中也无非就是这些内容，只不过界面组件更多一些罢了。

实例的代码如下：

（1）主程序中 WizardDialog 的使用。

```
//-----文件名：WizardDialog1.java----------
public class WizardDialog1 {
    public static void main(String[] args) {
```

```
        Shell shell = new Shell();
        WizardDialog dialog = new WizardDialog(shell, new MyWizard());
        dialog.setPageSize(-1, 60); //dialog 大小,-1 是指让宽度自动调整
        dialog.open();
    }
}
```

（2）WizardDialog 用到的 MyWizard 类的代码如下：

此类中主要是改写其父类 Wizard 的三种方法：addPages、canFinish、performFinish。在 addPages 方法中加入页面及为页面设置初始值，在 canFinish 方法中判断“完成”按钮是否可用，然后在 performFinish 方法中执行相应的处理程序。

```
//------文件名：MyWizard.java----------
public class MyWizard extends Wizard {
    /*
     * 把 NamePage 和 SchoolPage 两页面对象定义成实例变量，使用其他方法能访问得到
     */
    private NamePage namePage;//页面一，WizardPage 类
    private SchoolPage schoolPage;//页面二，WizardPage 类
    /**
     * 改写父类的 addPages 方法。在此方法将两个页面加入并设置
     */
    public void addPages() {
        //创建页面对象，并设置页面的名称
        namePage = new NamePage("namePage");
        schoolPage = new SchoolPage("schoolPage");
        //设置页面的初始值
        namePage.setName("陈刚");
        namePage.setEmail("glchengang@yeah.net");
        schoolPage.setSchool("广西师范大学");
        //加入 WizardPage，加入顺序决定了在界面显示的顺序
        addPage(namePage);
        addPage(schoolPage);
    }
    /**
     * 改写父类的 canFinish 方法，由此方法判断“完成”按钮何时可用、不可用
     * @return false“完成”按钮不可用，true 则相反
     */
    public boolean canFinish() {
        //设置为：当还没到最后一页时“完成”按钮不可用
        if (this.getContainer().getCurrentPage() != schoolPage)
            return false;
        return super.canFinish();
    }
    /*
     * 改写父类的 performFinish 方法，当单击“完成”按钮时，将执行此方法
```

```
     */
    public boolean performFinish() {
        //取出各页面的值
        System.out.println(namePage.getName());
        System.out.println(namePage.getEmail());
        System.out.println(schoolPage.getSchool());
        return true;
    }
}
```

（3）创建两个 WizardPage 的继承类：NamePage 页面、SchoolPage 页面。

NamePage 页面的代码和 16.4 节中扩展 Dialog 类或 TitleAreaDialog 类的代码在结构上非常相似：创建页面上的界面组件，添加组件的设值、取值方法。NamePage 代码如下：

```
//---------文件名：NamePage.java-----------
public class NamePage extends WizardPage {
    private Text nameText;
    private Text emailText;
    private String nameTextValue;
    private String emailTextValue;
    /**
     * 继承父类的构造函数
     */
    protected NamePage(String pageName) {
        super(pageName);
    }
    /*
     * 改写父类的 createControl 方法，在此方法中构建页面上的界面组件，注意不要在
     * 传入参数 parent 基础直接创建界面元素，而应在一个新面板 topComp 上创建
     */
    public void createControl(Composite parent) {
        //创建一个组件的底层面板，并使用 GridLayout 布局
        Composite topComp = new Composite(parent, SWT.NULL);
        topComp.setLayout(new GridLayout());
        //创建“姓名”文本框
        nameText = new Text(topComp, SWT.BORDER);
        nameText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        nameText.setText(nameTextValue == null ? "":nameTextValue);//防空值
        nameText.addModifyListener(new ModifyListener() {
            public void modifyText(ModifyEvent e) {
                dataChanged(); //输入值的合法性判断
            }
        });
        //创建“Email”文本框
        emailText = new Text(topComp, SWT.BORDER);
        emailText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
```

```
        emailText.setText(emailTextValue == null ? "" : emailTextValue);
        emailText.addModifyListener(new ModifyListener() {
            public void modifyText(ModifyEvent e) {
                dataChanged();
            }
        });
        //每页的提示信息
        setTitle("用户信息");
        setMessage("请输入您的姓名及 Email", INFORMATION);
        //必须要的一行
        this.setControl(topComp);
    }
    /**
     * 自定义方法，数据变化时对数据的合法性做检查
     */
    private void dataChanged() {
        if (nameText.getText().length() == 0) {
            setErrorMessage("姓名不能为空"); //提示错误信息
            setPageComplete(false); //使“完成”和“下一步”两按钮不可用
            return;
        }
        if (emailText.getText().length() > 0) {
            if (emailText.getText().indexOf("@") < 0) {
                setErrorMessage("Email 的格式错误");
                setPageComplete(false);
                return;
            }
        }
        setErrorMessage(null); //消除对话框上的出错提示文字
        setPageComplete(true); //使“完成”和“下一步”两按钮可用
    }
    /*
     * --------相应的 set/Get 方法 ----------
     */
    public String getName() {
        nameTextValue = nameText.getText();
        return nameTextValue;
    }
    public void setName(String string) {
        nameTextValue = string;
    }
    public String getEmail() {
        emailTextValue = emailText.getText();
        return emailTextValue;
    }
    public void setEmail(String string) {
```

```
        emailTextValue = string;
    }
}
```

第二个页面 SchoolPage 的代码与前页 NamePage 类似：

```
//---------文件名：SchoolPage.java-----------
public class SchoolPage extends WizardPage {
    private Text schoolText;
    private String schoolTextValue;
    protected SchoolPage(String pageName) {
        super(pageName);
    }
    public void createControl(Composite parent) {
        //创建一个组件的底层面板，并使用 GridLayout 布局
        Composite container = new Composite(parent, SWT.NULL);
        container.setLayout(new GridLayout());
        //创建“学校”文本框
        schoolText = new Text(container, SWT.BORDER);
        schoolText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
        schoolText.setText(schoolTextValue);
        schoolText.addModifyListener(new ModifyListener() {
            public void modifyText(ModifyEvent e) {
                dataChanged();//输入值的合法性判断
            }
        });
        //每页的提示信息
        setTitle("用户信息");
        setMessage("请输入您的学校的名称", INFORMATION);
        //必须要的一行
        this.setControl(container);
    }
    private void dataChanged() {
        if (schoolText.getText().length() == 0) {
            setErrorMessage("学校名不能为空");//提示错误信息
            setPageComplete(false);//使“完成”和“下一步”两按钮不可用
            return;
        }
        setErrorMessage(null); //消除对话框上的出错提示文字
        setPageComplete(true); //使“完成”和“下一步”两按钮可用
    }
    /*
     * --------相应的 set/get 方法 ----------
     */
    public String getSchool() {
        schoolTextValue = schoolText.getText();
```

```
        return schoolTextValue;
    }
    public void setSchool(String string) {
        schoolTextValue = string;
    }
}
```

16.2.3 向导式对话框使用的注意事项

向导式对话框在单击“下一步”、“上一步”按钮时都会执行 MyWizard 类的 canFinish 方法，因此在写这种方法时要注意保证程序的执行效率。

16.3 进度条对话框（ProgressMonitorDialog 类）

16.3.1 进度条对话框简介

使用较差的软件，常常有这种体验：单击了某按钮或菜单，系统却迟迟不见反应，鼠标失灵硬盘灯却在狂闪，死机了？有病毒？还是程序在执行超长任务？我们充满疑虑地等待了一会儿就再也受不了了，接下来就是关掉程序重启电脑，最后这个软件也被我们永远地踢出电脑。

在 Windows 操作系统中，最常用的进度条对话框就是文件复制时的弹出框。如果想让用户愉快地使用你开发的软件，那么在执行某个较长时间的操作时，就应该弹出一个进度条提示框，告诉用户程序正在做什么，做到什么地步了。进度条提示框可以提高用户的耐心。

第 11 章 SWT 组件的进度条 ProgressBar 类太底层，还需要处理诸如线程分配等问题。现在，JFace 提供了更易用的 ProgressMonitorDialog 类。

16.3.2 进度条对话框实例

在 Eclipse 开发一个进度条提示框是非常简单的，只要将执行代码写在某种方法中就行了。如图 16.25 所示，本例将在窗口中建立一个 GO 按钮，当单击该按钮时执行一个需时较长的操作（以一个每次停顿一秒的循环来模拟），这时会弹出如图 16.25 所示的一个进度条对话框。

此程序在未加进度条时的代码如下：

```
//-----文件名：ProgressMonitorDialog1.java----------
shell.setLayout(new RowLayout());
Button button = new Button(shell, SWT.BORDER);
button.setText("          GO          ");
```

```
button.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        //10 次循环，每次间隔一秒
        for (int i = 0; i < 10; i++) {
            try {Thread.sleep(1000);} catch (Throwable e2) {}
        }
    }
});
```

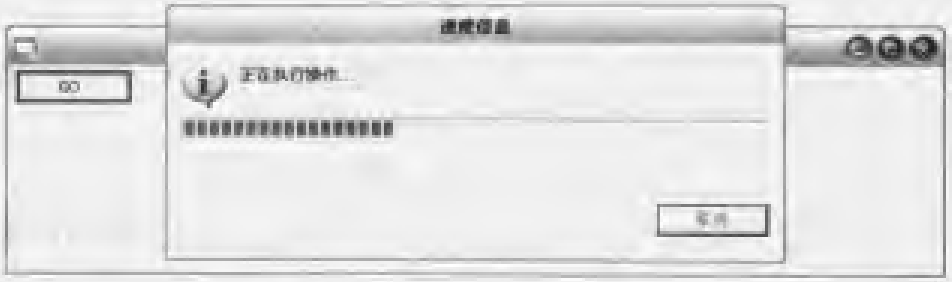


图 16.25　ProgressMonitorDialog 的效果图

如果希望在执行这段程序的时候弹出一个进度条，则共要作如下三步修改：

- 创建一个 ProgressMonitorDialog 对象。
- 编写一个处理过程接口 IRunnableWithProgress 的匿名实现类，并创建其对象，将原代码嵌入到其 run 方法中。当然这个类不一定要写成匿名类，写成命名内部类或外部类也可以。
- 用 ProgressMonitorDialog 对象的 run 方法接收 IRunnableWithProgress 对象做参数，并弹出进度条对话框，run 这一条语句要用 try/catch 括起来。

完整的程序代码如下：

```
shell.setLayout(new RowLayout());
Button button = new Button(shell, SWT.BORDER);
button.setText("          GO          ");
button.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        //第一步：创建一个进度条对话框对象
        ProgressMonitorDialog pmd = new ProgressMonitorDialog(shell);
        //第二步：创建进度条对话框的处理过程对象
        IRunnableWithProgress rwp = new IRunnableWithProgress() {
            public void run(IProgressMonitor monitor) throws InvocationTargetException,
InterruptedException {
                monitor.beginTask("开始执行.....", 10);
                //10 次循环，每次间隔一秒
                for (int i = 0; i < 10; i++) {
                    try {
                        Thread.sleep(1000);
                    } catch (Throwable e2) {} //间隔一秒
                    monitor.setTaskName("第" + (i + 1) + "次循环");
```

```
                monitor.worked(1);//进度条前进一步
            }
            monitor.done();//进度条前进到底
        }
    };
    try {
        //第三步：运行对话框，第二个参数为 false 表示对话框上的取消按钮不可用
        pmd.run(false, false, rwp);
    } catch (Exception e3) {
        e3.printStackTrace();
    }
  }
});
```

程序说明：

monitor.beginTask("开始执行.....", 10);一句中 10 表示将进度条分成 10 格。如果无法确定间隔数，可以将该值设得大一些，例如设成 20，即使真实循环数没有达到 20 次或超过了 20 次也不会引起程序出错。

IRunnableWithProgress 是一个接口，本例是用匿名内部类的写法。进度条对话框的代码看似复杂，其实简单而无变化，基本都分为上面所说的三步，而原有的处理代码则包裹在 IRunnableWithProgress 实现类的 run 方法中。总结它的代码框架都是一个样了，只要套用此框架就行了，框架代码如下：

```
try {
    ProgressMonitorDialog pmd = new ProgressMonitorDialog(shell);
    IRunnableWithProgress rwp = new IRunnableWithProgress() {
        public void run(IProgressMonitor monitor) throws InvocationTargetException,
InterruptedException {
            ..... （原有执行代码）
        }
    };
    pmd.run(false, false, rwp);
} catch (Exception e3) {
    e3.printStackTrace();
}
```

16.4 其他类型对话框

在 SWT 中还有一些对话框在开发系统时也是常会用到的，集中在此一起介绍。

16.4.1 信息提示框（MessageBox 类）

如果仅提示很少的信息，则 JFace 的 MessageDialog 弹出的对话框太大了，SWT 的

MessageBox 也许更好一些。两者相比较而言，MessageBox 可以自动调整窗口大小，但弹出一个对话框窗口 MessageBox 要三句，MessageDialog 只要一句。

1．MessageBox 最简单的使用代码

```
//--------文件名：MessageBox1.java --------
MessageBox messageBox = new MessageBox(shell);
messageBox.setMessage("Hello World!");
messageBox.open();
```

效果如图 16.26 所示。

2．用到 MessageBox 的式样参数和 open 方法返回值实例

```
//--------文件名：MessageBox2.java --------
MessageBox msgbox= new MessageBox(shell,SWT.ICON_QUESTIONISWT.YESISWT.NO);
msgbox.setMessage("确认删除吗？ ");
int rc = msgbox.open();
if (rc == SWT.YES)
    System.out.println("您单击了 "是" 按钮");
if (rc == SWT.NO)
    System.out.println("您单击了 "否" 按钮");
```

代码中使用式样 SWT.ICON_QUESTION 来显示问号图标、式样 SWT.YES | SWT.NO 来显示“是、否”按钮，并根据用户回答执行不同的打印语句，其界面效果如图 16.27 所示。

图 16.26　MessageBox 实例 1

图 16.27　MessageBox 实例 2

3．MessageBox 的图标式样（如图 16.28 所示）

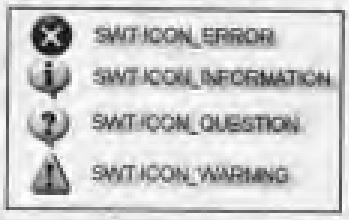


图 16.28　MessageBox 图标式样的对照图

4．MessageBox 的按钮式样（如表 16.1 所示）

表 16.1　MessageBox的按钮式样

按钮式样组合	显示的按钮
SWT.OK	确认
SWT.OK	SWT.CANCEL	确认、取消

续表

按钮式样组合	显示的按钮
SWT.YES | SWT.NO	是、否
SWT.YES | SWT.NO | SWT.CANCEL	是、否、取消
SWT.RETRY | SWT.CANCEL	重试、取消
SWT.ABORT | SWT.RETRY | SWT.IGNORE	终止、重试、忽略

16.4.2　颜色选择对话框（ColorDialog 类）

SWT/JFace 中设置颜色都要用到 Color 类，如 button.setBackground（Color color）是设置按钮背景色。而 ColorDialog 则可以弹出一个颜色选择对话框给用户选择颜色，如图 16.29 所示。

图 16.29　颜色选择对话框

使用 ColorDialog 的实例如下：

```
//--------文件名：ColorDialog1.java --------
//创建颜色选择对话框
ColorDialog dlg = new ColorDialog(shell);
//打开颜色选择对话框，并得到包含所选颜色的 RGB 值的对象
RGB rgb = dlg.open();
//根据 rgb 生成一个 color 对象
if (rgb != null) {
    Color color = new Color(shell.getDisplay(), rgb);
    // 使用颜色对象 color  ……
    // 一定要记得用完后，调用 color.dispose()将 color 释放掉
}
```

16.4.3　字体选择对话框（FontDialog 类）

FontDialog 提供了字体选择对话框，如图 16.30 所示，它返回一个字体数据对象

FontData，然后可以用 FontData 生成一个字体对象 Font。

图 16.30　字体选择对话框

使用 FontDialog 的实例如下：

```
//--------文件名：FontDialog1.java --------
FontDialog dlg = new FontDialog(shell);
FontData fontData = dlg.open();
if (fontData != null) {
   Font font = new Font(shell.getDisplay(), fontData);
   // 使用字体对象 font ……
   // 一定要记得用完后，调用 font.dispose()将 font 释放掉
}
```

16.4.4　打印设置对话框（PrintDialog 类）

打印在软件系统中是必不可少的功能，在 SWT/JFace 中要打印就要用到打印类 Printer，PrintDialog 则提供了打印设置对话话框，如图 16.31 所示，它可以返回一个打印数据对象 PrinterData，然后根据 PrinterData 可以得到打印对象 Printer。

图 16.31　打印设置对话框

使用 PrintDialog 的实例如下：

```
//--------文件名：PrintDialog1.java --------
PrintDialog dlg = new PrintDialog(shell);
PrinterData printerData = dlg.open();
if (printerData != null) {
   Printer printer = new Printer(printerData);
   //使用 printer 对象
   //一定要记得用完后，调用 printer.dispose()将 printer 释放掉
}
```

以上程序只弹出一个打印设置对话框，然后返回一个 Printer 对象，那么如何使用 Printer 对象呢，这就没有前面的 Color、Font 简单了。以下程序给出了一个如何打印一个文本文件的方法：

```
//-----------文件名：PrintTextExample.java-----------
public class PrintTextExample {
    public static void main(String[] args) {
        try {
            PrintTextExample window = new PrintTextExample();
            window.open();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void open() {
        final Display display = Display.getDefault();
        final Shell shell = new Shell();
        //用文件选择对话框取得一个文件名字符串
        FileDialog fileDlg = new FileDialog(shell, SWT.OPEN);
        String fileName = fileDlg.open();
        if (fileName != null) {
            //用打印对话框取得一个打印对话 printer
            PrintDialog printDlg = new PrintDialog(shell);
            PrinterData printerData = printDlg.open();
            if (printerData != null) {
                Printer printer = new Printer(printerData);
                //打印文件
                try {
                    String contents = getFileContents(fileName);//取出文件内容
                    MyPrinter p = new MyPrinter(printer, fileName, contents);
                    p.print();
                } catch (java.lang.Exception e) {
                    e.printStackTrace();
                }
                printer.dispose();//printer 对象没用后要手动 dispose 掉
```

```
            }
        }
        display.dispose();
    }
    /**
     * 取出文件内容的方法
     */
    private String getFileContents(String fileName) throws FileNotFoundException, IOException {
        StringBuffer contents = new StringBuffer();
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader(fileName));
            while (reader.ready()) {
                contents.append(reader.readLine());
                contents.append("\n"); //每读完一行，换行一次
            }
        } finally {
            if (reader != null)
                try {
                    reader.close();
                } catch (IOException e) {}
        }
        return contents.toString();
    }
    /**
     * 自定义的打印实现类，将所需参数传入，调用 print 方法即可打印
     */
    private class MyPrinter {
        private Printer printer; // 打印对象
        private String fileName; //文件名（带路径）
        private String contents; //文件名对应的文件内容
        private GC gc; //一个 GC 对象
        private int xPos, yPos; //打印对象 printer 用的坐标
        private Rectangle bounds; //打印空间
        private StringBuffer buf; // Holds a word at a time
        private int lineHeight; // The height of a line of text

        /**
         * 构造函数
         */
        public MyPrinter(Printer printer,String fileName,String contents){
            this.printer = printer;
            this.fileName = fileName;
            this.contents = contents;
        }
        /**
```

```
     * 打印文件的方法
     */
    public void print() {
        if (printer.startJob(fileName)) {//开始打印任务
            //设定打印空间
            bounds = computePrintArea(printer);
            xPos = bounds.x;
            yPos = bounds.y;
            //创建 GC 对象
            gc = new GC(printer);
            //设定线的高度
            lineHeight = gc.getFontMetrics().getHeight();
            //设定 tab 键的空格数
            int tabWidth = gc.stringExtent(" ").x;
            //开始打印
            printer.startPage();
            buf = new StringBuffer();
            char c;
            for (int i = 0, n = contents.length(); i < n; i++) {
                //得到文件内容的字符
                c = contents.charAt(i);
                //如果读到\n，调用 printBuffer 方法将 buf 中的字符打印，并换行
                if (c == '\n') {
                    printBuffer();
                    printNewline();
                }
                //如果读到\t，表示要跳过一定的空格
                else if (c == '\t') {
                    xPos += tabWidth;
                } else {
                    buf.append(c);//将字符添加进 buf 变量
                    //检查是否有空间，如果到打印
                    if (Character.isWhitespace(c))
                        printBuffer();
                }
            }
            printer.endPage();
            printer.endJob();
            gc.dispose();
        }
    }
    /**
     * 打印缓存 buf 变量中的字符
     */
    private void printBuffer() {
        //取得缓存中的字符宽度
```

```
        int width = gc.stringExtent(buf.toString()).x;
        //如果宽度不够，则换行
        if (xPos + width > bounds.x + bounds.width)
            printNewline();
        //打印缓存 buf 变量中的字符
        gc.drawString(buf.toString(), xPos, yPos, false);
        xPos += width;
        buf.setLength(0);
    }
    /**
     * 打印换行的方法
     */
    private void printNewline() {
        //重设新行的坐标
        xPos = bounds.x;
        yPos += lineHeight;
        //如果超过页长度，则换一页打印
        if (yPos > bounds.y + bounds.height) {
            yPos = bounds.y;
            printer.endPage();
            printer.startPage();
        }
    }
    /**
     * 计算打印空间的方法
     */
    private Rectangle computePrintArea(Printer printer) {
        //取得打印空间
        Rectangle rect = printer.getClientArea();
        Rectangle trim = printer.computeTrim(0, 0, 0, 0);
        //取得打印机的 DPI（DPI：表示每英寸的点数,即通常说的打印机的分辨率）
        Point dpi = printer.getDPI();
        //计算可打印空间
        int left = trim.x + dpi.x;
        if (left < rect.x)
            left = rect.x;
        int right = (rect.width + trim.x + trim.width) - dpi.x;
        if (right > rect.width)
            right = rect.width;
        int top = trim.y + dpi.y;
        if (top < rect.y)
            top = rect.y;
        int bottom = (rect.height + trim.y + trim.height) - dpi.y;
        if (bottom > rect.height)
            bottom = rect.height;
        return new Rectangle(left, top, right - left, bottom - top);
```

```
        }
    }
}
```

16.4.5 目录选择对话框（DirectoryDialog 类）

目录选择对话框如图 16.32 所示，它提供一个界面让用户选择目录，然后返回所选目录的路径（一个字符串）。

图 16.32 目录选择对话框

使用 DirectoryDialog 的实例如下：

```
//--------文件名：DirectoryDialog1.java --------
DirectoryDialog dlg = new DirectoryDialog(shell);
dlg.setText("目录");  //设置窗口标题
dlg.setMessage("请选择一个目录："); //设置提示文字
dlg.setFilterPath("c:/windows");  //设置初始目录
String dir = dlg.open(); //打开对话框并返回一个包含所选目录路径的字符串
if (dir != null)
    System.out.println(dir); //例如选择“我的文档”，则会打印“D:\My Documents”
```

16.4.6 文件选择对话框（FileDialog 类）

文件选择对话框如图 16.33 所示，它提供一个界面让用户选择文件，然后返回所选文件的全路径（路径+文件名）。FileDialog 还能设置文件选择的类型限制，也能选择多个文件。

1．一个最简单的实例

```
//--------文件名：FileDialog1.java --------
FileDialog dlg = new FileDialog(shell, SWT.OPEN);
dlg.setFilterPath("c:/windows"); //设置初始路径
String fileName = dlg.open(); //返回所选文件的全路径（路径＋文件名）
```

效果如图 16.33 所示。

图 16.33　文件选择对话框

2．选择多个文件（SWT.MULTI 式样）

```
//--------文件名：FileDialog2.java --------
FileDialog dlg = new FileDialog(shell, SWT.OPEN | SWT.MULTI);//加 MULTI 式样
String fileName = dlg.open(); //返回最后一个选择文件的全路径
String[] fileNames = dlg.getFileNames(); //返回所有选择文件的文件名，不包含路径
System.out.println(fileName == null ? "" : fileName);//打印 open 方法的返回值
//用一个循环将数组中的文件名加上路径打印出来
for (int i = 0; i < fileNames.length; i++)
    System.out.println(dlg.getFilterPath() + "\\" + fileNames[i]);
```

3．文件选择的限制类型（setFilterExtensions 方法）

```
//--------文件名：FileDialog3.java --------
FileDialog dlg = new FileDialog(shell, SWT.OPEN | SWT.MULTI);
dlg.setFilterNames(new String[] { "可执行文件 (*.exe)", "Microsoft Excel (*.xls)"});
dlg.setFilterExtensions(new String[] { "*.exe", "*.xls", "*.jpg", "*.*" });
String fileName = dlg.open();
```

程序说明：

- setFilterExtensions 设置文件选择的限制类型。
- setFilterNames 设置文件类型的说明文字，它不要也可以。
- 如图 16.34 所示，setFilterExtensions 参数的数组元素会和 setFilterNames 参数的数组元素依顺序相对应，像*.jpg 类型本应该对应 setFilterNames 中数组的第三个元素，但此数组只有两个元素，所以 setFilterNames 没有为*.jpg 类型设置名称，则界面默认以类型*.jpg 作为其名称。

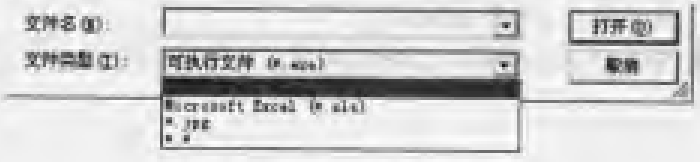


图 16.34　限制文件选择的类型

4. 保存对话框（SWT.SAVE 式样）

保存对话框并不能提供实际保存文件的功能，它仅仅是弹出一个文件对话框，然后返回一个包含路径信息的字符串，具体文件的保存操作（指将文件信息写入磁盘）需要另外写程序去实现。

保存对话框的实例如下：

```
//--------文件名：FileDialog4.java --------
FileDialog dlg = new FileDialog(shell, SWT.SAVE);
String fileName = dlg.open();
if (fileName != null)
    System.out.println(fileName);
```

16.5 本 章 小 结

本章着重介绍了 JFace 中各类对话框，本章之后，SWT/JFace 的内容就介绍完了，但 SWT/JFace 中的组件还有很多。根据笔者的实践经验，如果是开发数据库应用型软件，本书介绍组件已经足以完成 95%以上的界面开发。如果需要自制组件，就要用到 Draw2D 的知识，这已经超出了本书的内容范围及笔者所掌握的知识，关于 Draw2D 仅在 www.eclipse.org 上有数篇文章提及。

第 3 篇

插件开发篇

第 17 章　Eclipse 插件开发起步

第 18 章　常用插件扩展点

第 19 章　Eclipse 插件的国际化

第 20 章　报表：用 POI 与 Excel 交互

第 21 章　项目打包与发行

第 22 章　插件项目实战

第 17 章　Eclipse 插件开发起步

17.1　Eclipse 插件开发概述

17.1.1　插件开发简介

插件的概念大家应该很熟悉，像 MP3 播放软件 WINAMP 的皮肤插件、Windows Media Player 的众多的外观插件、音效插件等。如果你以为插件只能做成主软件的边角料，那么 Eclipse 插件应该可以更新你对插件的看法。

Eclipse 中有三个最吸引人的地方：一是它创新性的图形 API，即 SWT/JFace；二是它的插件机制；三是利用它的插件机制开发的众多功能强大的插件。Eclipse 也是对插件概念发挥得最炉火纯青的一个软件，完全可以这么说，插件成就了 Eclipse 的今天。

Eclipse 的内核很小，其他功能都是基于这个内核上的插件，如 Eclipse 自带的 JUnit、Ant 等。而且 Eclipse 还开放了自己的插件机制，并提供了很好的插件开发环境，让用户可以轻松开发 Eclipse 插件。Eclipse 的插件能达到什么程度？看看这些 Eclipse 上流行的插件吧，用于 UML 建模的 Together for Eclipse、用于 J2EE 的 Lomboz 和 MyEclipse、IBM 的 Java 开发工具 WSAD 等，它们全是 Eclipse 的插件。如果微软愿意，它也可以把 Office 软件做成 Eclipse 的插件，如果 Adobe 有兴趣，Photoshop 也可以有 for Eclipse 的插件版，Eclipse 中的绘图 API－Draw2D 的功能也是很强的。

Eclipse 的各式插件正如雨后春笋般不断冒出，Eclipse 本身已经超越了开发环境的概念，而将成为一个通用的平台，让尽量多的软件作为插件集成在上面，成为未来的集成桌面环境。同样也可以将我们的应用系统写成 Eclipse 插件。笔者就在 2004 年参与开发了一个项目管理软件，该软件就是以 Eclipse 的插件形式开发的。

17.1.2　Eclipse 插件开发的优势和不足

那么把软件写成 Eclipse 插件有什么好处呢？对于用户来说 Eclipse 的使用环境比较友好，前面介绍的 SWT/JFace 中还是比较基本的界面元素，像 Eclipse 中的视图、编辑窗、停泊窗这些界面如果用 Appliction 的方式会很麻烦，但用 Eclipse 插件来实现，则不费吹灰之力。因此， Eclipse 插件的最大优点就是界面友好、风格统一，特别是对于熟悉 Eclipse 操作的用户，这种优势就更明显。

当然将软件写成插件形式也有一定的缺陷。首先，插件必须依附 Eclipse 运行，如果要安装插件就得先安装 Eclipse；其次，插件和 Eclipse 融合在一起，原 Eclipse 的一些菜单和

工具栏是无法完全屏蔽的。

17.2　插件的 Hello World

17.2.1　使用向导一步步创建 HelloWorld

本小节利用 Eclipse 的“新建”向导来创建一个简单的插件。

1．新建一个插件项目

（1）选择主菜单“文件→新建→项目”选项，在弹出的窗口中（如图 17.1 所示）选择“插件开发→插件项目”选项，然后单击“下一步”按钮。

图 17.1　项目类型选择

（2）如图 17.2 所示，输入项目名 myplugin，其他设置不变，然后单击“下一步”按钮。

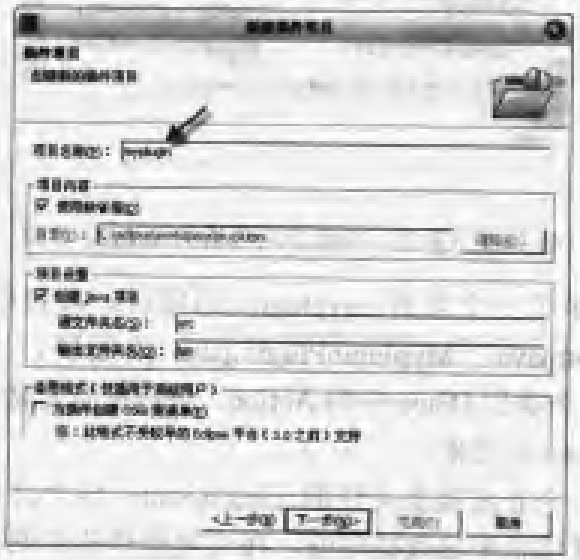


图 17.2　项目名称

（3）在新显示的窗口中接受所有默认值不变，直接单击“下一步”按钮，这时将显示模板选择窗口（如图 17.3 所示）。选中“使用其中一个模板来创建插件”选项，然后选择模板 Hello,World 选项。最后单击“完成”按钮结束向导对话框。

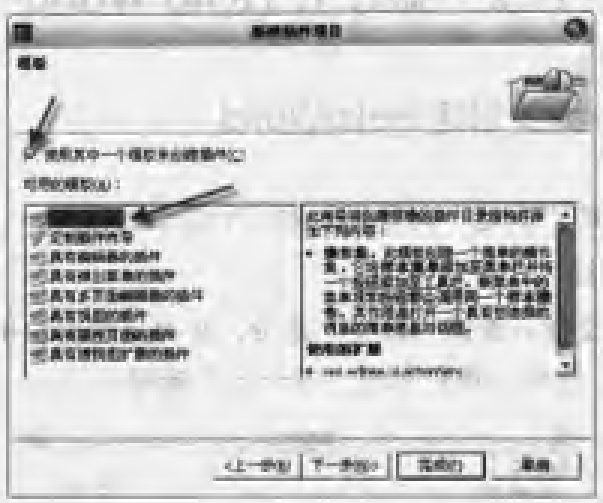


图 17.3　模板选择窗口

2．插件项目 myplugin 简介

如果在新建项目中操作正确，Eclipse 将显示如图 17.4 所示的界面。

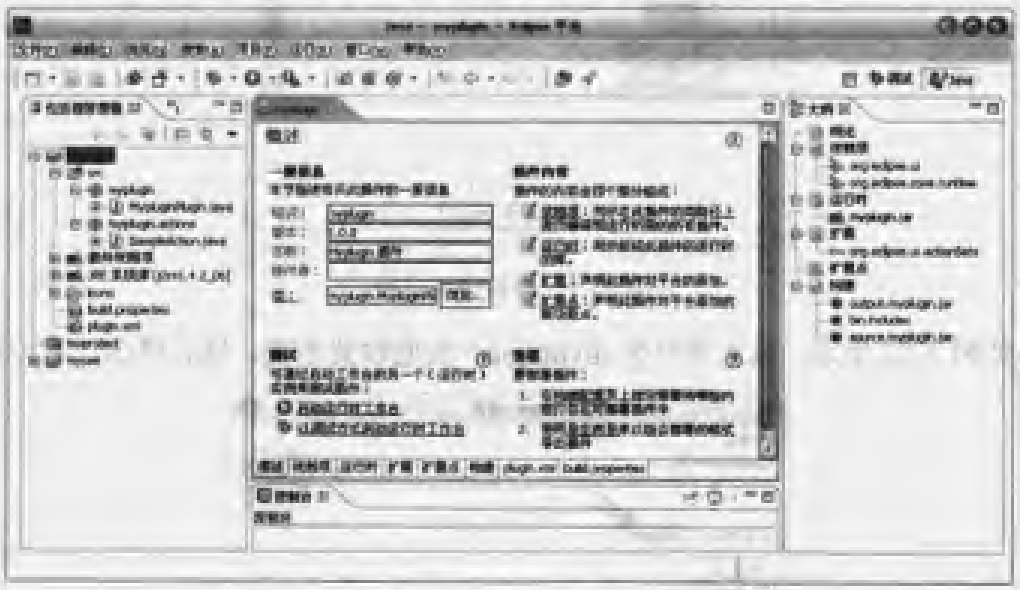


图 17.4　建立一个插件项目后的 Eclipse 界面

界面的左边视图中多了一个名为 myplugin 的项目。项目中有两个文件：Myplugin Plugin.java、SampleAction.java。MypluginPlugin.java 较重要，今后将会使用到它，而 SampleAction.java 则是一个类似 JFace 中的 Action，可以把它看做是插件中的 Action。运行时将看到 SampleAction.java 的效果。

项目根目录下还有一个非常重要文件的 plugin.xml，这个文件是插件的入口，Eclipse 是根据这个文件里的设置信息来加载插件的。在开发插件的初期会频繁地在这个文件中做编辑，术语叫“设置扩展点”。像在 Eclipse 的增加主菜单、视图、按钮等，都是在这个文

件里面设置不同的扩展点，后面将详细讲到如何编辑此文件。

也许有人会问：开发一个系统会有很多的菜单和按钮，是不是都要在 plugin.xml 文件里设置呢？回答：不必。在 plugin.xml 里只设置嵌入到 Eclipse 框架里的主要扩展点，其他软件自有的菜单和按钮就不用在 plugin.xml 里设置了。

如图 17.4 所示的 Eclipse 界面中部显示的就是 plugin.xml 的编辑窗口。单击该窗口下部（见图 17.5）的 plugin.xml 项，可以直接编辑此文件。

图 17.5　plugin.xml 编辑窗下部的选项条

3．运行插件

如图 17.6 所示，选择主菜单“运行→运行方式→运行工作平台”选项，这种是专用于插件的运行方式，它将打开一个新的 Eclipse 环境，并同时将插件项目编译加载到新 Eclipse 环境中。今后开发经常要通过这种方法来试运行所开发的插件项目，不过那时候选择“运行→调试方式→运行工作平台”更适合调试。Eclipse 支持调试期间的部分热修改，不用每次修改都新启动一个 Eclipse，这样能节省很多调试开发时间。

新开的 Eclipse 界面如图 17.6 所示。在新的 Eclipse 环境中新增加了一个工具栏按钮和一个主菜单项。单击此按钮或菜单项，将弹出一个“Hello，Eclipse world”信息提示框。按钮和菜单项都是由 SampleAction.java 产生的效果。

图 17.6　myplugin 插件运行效果图

4．总结

本小节里还只是依样画葫芦，读者感觉有点云里雾里的吧。但不管怎么样，第一个 Eclipse 插件已经在我们手里诞生了。17.2.2 小节将不使用 HelloWorld 模板，而是新建一个空白的插件项目，然后一步步设置加编程来实现这个 Hello World 插件项目。

17.2.2　以空白项目为基础创建 HelloWorld

1．新建项目

按照 17.2.1 小节所讲新建插件项目的方法，新建一个名为 myplugin2 的插件项目。注意

在最后一步不要选择任何模板，直接单击“完成”按钮结束向导对话框，除此之外的其他步骤都一样。很幸运，Eclipse 3.0 修正了这里的 BUG，以前用 Eclipse 2.x 加多国语言包时，在这一步还会出现很多库引用的错误，要手工一个个去修正。

2．创建 IWorkbenchWindowActionDelegate 接口的实现类

新建一个包 cn.com.chengang，并将 17.2.1 小节中由 HelloWorld 模板生成的 myplugin 项目中的 SampleAction.java 文件复制到本项目中（Eclipse 支持鼠标拖动操作）。然后对 SampleAction 作了一些小修改，删除了无用的注释和构造函数，修改了一下弹出框的提示文字。修改后的代码如下：

```
/**
 * 本类相当于插件的 Action，要在 Eclipse 中增加主菜单或工具栏按钮，
 * 就需要写一个实现 IWorkbenchWindowActionDelegate 接口的类
 */
public class SampleAction implements IWorkbenchWindowActionDelegate {
    private IWorkbenchWindow window;
    public void run(IAction action) {
        //打开一个信息提示框
        MessageDialog.openInformation(window.getShell(),
                                      "Myplugin2 插件", "Hello，这是手工做的插件");
    }
    public void selectionChanged(IAction action, ISelection selection) {}
    public void dispose() {}
    public void init(IWorkbenchWindow window) {this.window = window;}
}
```

3．原 plugin.xml 文件各设置项说明

如图 17.7 所示，将 plugin.xml 文件打开，并单击窗口下的 plugin.xml 转到代码编辑窗。

图 17.7　plugin.xml 的代码编辑窗

详细介绍其中的各项设置如下：

（1）<plugin>项

```
<plugin
   id="myplugin2"
   name="Myplugin2 插件"
   version="1.0.0"
   provider-name=""
   class="myplugin2.Myplugin2Plugin">
```

说明：<plugin>是 plugin.xml 的主体。

- id：插件的惟一标识。实际项目中一般加上包名或网址名来命名 id，例如 eclipse 的 tomcat 插件是这样命名的：org.eclipse.tomcat，这样在世界上就不会有插件的标识名和你重名了。以后在某些扩展点中的属性也会用到标识符作为名称的前缀。
- name：插件的名称，可以不惟一。
- version：插件版本号。
- provider-name：插件开发商的名称，可以写上作者姓名或公司名称。
- class：插件类的名称，即插件项目自动生成的 MypluginPlugin2.java 文件的类，前面加上包名。

（2）< runtime>项

```
<runtime>
  <library name="myplugin2.jar">
      <export name="*"/>
  </library>
</runtime>
```

说明：这里是声明插件运行时需要的 jar 包。例如，插件要连接 MySQL 数据库时需要它的一个包，则定义如下，其中“lib\”是该包所在路径，本插件自身的 jar 包也要声明，而且本插件在打包时，将以 myplugin2.jar 为名打包。

```
<runtime>
  <library name="myplugin2.jar">
      <export name="*"/>
  </library>
  <library name="lib\mysql-connector-java-3.0.9-stable-bin.jar"/>
</runtime>
```

（3）<requires>项

```
<requires>
  <import plugin="org.eclipse.ui"/>
  <import plugin="org.eclipse.core.runtime"/>
</requires>
```

说明：在 requires 域中定义了该插件所要使用的依赖插件。现在有两项就够了，随着开发的不断深入这里将会添加更多对其他插件的引用。如下代码是笔者的实际项目中的 requires 设置，它要用到 draw2d 和 gef 插件来画图，用于插件的帮助系统来创建自己的帮助

文档。

```
<requires>
    <import plugin="org.eclipse.ui"/>
    <import plugin="org.eclipse.core.runtime"/>
    <import plugin="org.eclipse.core.resources"/>
    <import plugin="org.eclipse.draw2d"/>
    <import plugin="org.eclipse.gef"/>
    <import plugin="org.eclipse.help"/>
    <import plugin="org.eclipse.help.ui"/>
    <import plugin="org.eclipse.help.appserver"/>
    <import plugin="org.eclipse.help.webapp"/>
</requires>
```

4. 为 HelloWorld 修改 plugin.xml

将如下代码加入到 plugin.xml 的</requires>行之后：

```
<extension point="org.eclipse.ui.actionSets">
    <actionSet
        label="样本操作集"
        visible="true"
        id="myplugin2.actionSet">
      <menu
          label="样本菜单(&amp;M)"
          id="sampleMenu">
        <separator
            name="sampleGroup">
        </separator>
      </menu>
      <action
          label="样本操作(&amp;S)"
          icon="icons/sample.gif"
          class="cn.com.chengang.SampleAction"
          tooltip="Hello，这是手工做的插件"
          menubarPath="sampleMenu/sampleGroup"
          toolbarPath="sampleGroup"
          id="cn.com.chengang.SampleAction">
      </action>
    </actionSet>
</extension>
```

说明：

在<extension>项设置要扩展的扩展点，它是非常重要的一项。

- point="org.eclipse.ui.actionSets"，设置了本插件的扩展点，actionSets 是指 Eclipse 的菜单、菜单项和工具栏按钮的扩展点。
- <actionSet>项表示一个 action 组（菜单、按钮）。label 是显示的名称。id 其惟一标识符，只要保证在本 plugin.xml 文件中不存在重复的 id 就行了。visible 指设置的按钮或菜单是否显示，如果设置成 false，则不显示。（注意：看 visible 设置的效果，要先将“透视图”关掉再重新打开。）

- <menu>是<actionSet>下的子项，它表示在 Eclipse 中插入显示一个名为“样本菜单(M)”的主菜单。separator 标签是一个结束符，它可以对菜单分组。
- <action>也是<actionSet>下的子项，由它设置菜单、按钮。icon 是图片的路径，如果该图片不存在，默认是不显示图片而显示文字。Class 是按钮所对应的类，注意，包名也要加上。menubarPath 表示把这个 action 做成一个菜单项，放在上面<menu>定义的主菜单下。toolbarPath 表示把这个 action 再做成一个工具栏按钮。id 是标识符，建议设置成和 class 项一样的名称。

以上仅是 Eclipse 的扩展点中的一种，此外还有其他的扩展点共有一百多种之多。我们没有必要了解所有扩展点的设置，只须熟悉一些常用的扩展点即可，如视图的扩展点 org.eclipse.ui.views、编辑器的扩展点 org.eclipse.ui.editors 等，后面将陆续给予介绍。另外，各种扩展点在 Eclipse 的帮助中有详细的说明，其位置为：选择主菜单“帮助→帮助内容”选项，然后打开“平台插件开发指南→参考→扩展点参考”选项。

5. 运行插件

先将 17.2.1 小节所建的 myplugin 项目关闭掉，以免干扰我们的视线。关闭方法：右键单击 myplugin 项目名，在弹出菜单中选择“关闭项目”选项，然后再运行插件 myplugin2 插件，效果如图 17.8 所示。

图 17.8　myplugin2 插件运行效果图

注意：

（1）如果在插件开发出现错误，错误信息会显示在新建的 Eclipse 的“错误日志”视图，该视图在“显示视图”窗口中的“PDE 运行时”视图组下。

（2）由于运行插件较为费时，而 Application 方式则小巧快速。所以一般在开发中先写一个简单的 Appliation 来调用和测试插件中的功能部件。例如：要在插件中写一个自定义的对话框，那么先写一个调用此对话框的 Application，当初步完成后，再在插件中使用该对话框。

17.3　本章小结

本章主要介绍了如何创建一个简单的插件，插件虽然简单，但其中却包含了插件相关的主要内容。本例最重要的就是 plugin.xml 文件的设置，掌握它是掌握插件开发的必经之路。

第 18 章　常用插件扩展点

在第 17 章对 plugin.xml 作了少量介绍，plugin.xml 是插件和 Eclipse 内核的接口，Eclipse 就像一所大宅子，它的外墙（plugin.xml）有很多的门（扩展点），要熟练进出这座大宅子，先得搞清楚它有哪些门（扩展点）。

插件的扩展点非常之多，但很多扩展点都不常用到，只要熟悉一些主要的扩展点即可。本节将面向实际开发需要来介绍这些扩展点，并且本章所有实例都在第 17 章建立的 myplugin2 插件项目的基础上创建。

18.1　加入透视图（perspectives）

开发一个插件，最常用的方式就是新增一个属于本插件专有的透视图，然后在此透视图基础上来展开功能，本书也采用这种方式。

18.1.1　准备工作

先将以前用到的包括图标的 icons 目录复制一份到 myplugin2 项目中，复制后的路径如图 18.1 所示。

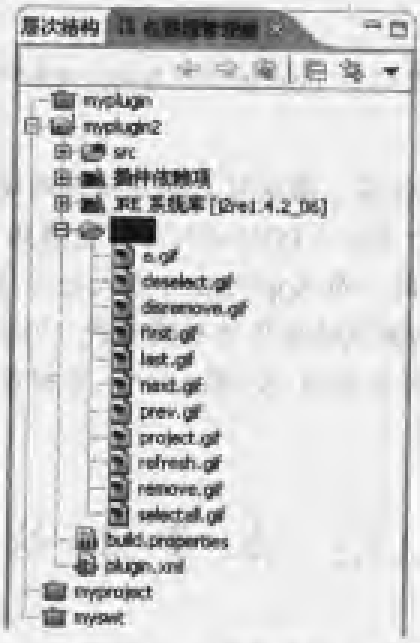


图 18.1　图标的路径

18.1.2　修改 plugin.xml 文件，设置透视图的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前：

```
<extension
      point="org.eclipse.ui.perspectives">
  <perspective
        name="myplugin 透视图"
        icon="icons/selectall.gif"
        class="cn.com.chengang.SamplePerspective"
        id="cn.com.chengang.SamplePerspective">
  </perspective>
</extension>
```

代码说明：

- org.eclipse.ui.perspectives 是透视图的扩展点。
- name：透视图的名称。
- icon：透视图的图标。
- class：透视图所对应的类（还没编写，下一步将完成此类）。
- id：透视图标识，建议设置成和 class 一样的名称，省得以后扩展点设置得太多，让人糊涂。

18.1.3　建立透视图类

在 18.1.2 小节的 plugin.xml 中提前设置了透视图对应的类 cn.com.chengang.SamplePerspective，这一步就在包 cn.com.chengang 中创建此类。透视图类必须实现 IperspectiveFactory 接口，此接口只有一种方法 createInitialLayout，先让它空实现。

SamplePerspective 类的代码如下：

```
//--------文件名：SamplePerspective.java----------------
public class SamplePerspective implements IPerspectiveFactory {
    public void createInitialLayout(IPageLayout layout) {}
}
```

18.1.4　运行插件

运行插件，然后在新 Eclipse 环境中选择主菜单“窗口→打开透视图→其他”选项。在弹出窗口中，可以看到一个名为 myplugin 透视图的项，如图 18.2 所示。

选择并打开“myplugin 透视图”选项后，显示如图 18.3 所示的 Eclipse 界面。我们发现该透视图光秃秃的什么也没有。没关系，下面就会向这个透视图中加入两个视图。

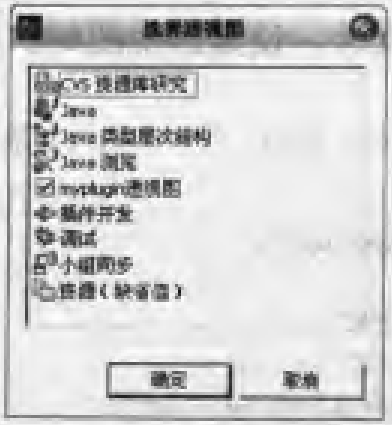


图 18.2　选择透视图

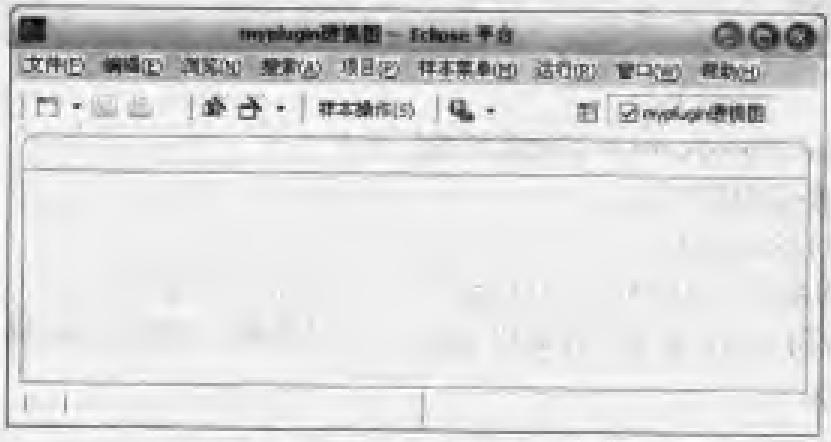


图 18.3　myplugin 透视图的效果图

18.1.5　总结

由本节可以看到，在 Eclipse 插件环境中，创建一个菜单、按钮、透视图界面是多么简单，都不用编写实际界面的创建代码，只要设置一些扩展点就行了。

18.2　在透视图中加入视图（views）

接着 18.1 节的内容，给透视图加入两个视图，实现的步骤如下所述。

18.2.1　修改 plugin.xml 文件，设置视图的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>之前：

```
<extension
      point="org.eclipse.ui.views">
   <category
         name="myplugin2 视图"
```

```
        id="com.glchengang.myplugin2.view">
  </category>
  <view
        name="视图 1"
        icon="icons/prev.gif"
        category="com.glchengang.myplugin2.view"
        class="cn.com.chengang.View1"
        id="cn.com.chengang.View1">
  </view>
  <view
        name="视图 2"
        icon="icons/project.gif"
        category="com.glchengang.myplugin2.view"
        class="cn.com.chengang.View2"
        id="cn.com.chengang.View2">
  </view>
</extension>
```

代码说明：

- org.eclipse.ui.views 是视图的扩展点。
- <category>…</category>是视图的分组名及 id 标识，它的效果体现在“显示视图”窗口里，显示视图的打开方法是：在主菜单选择“窗口→显示视图→其他”选项，图 18.4 中的左图是本例设置的效果。

  <category>项中的 id 属性要保证它在 Eclipse 的所有插件中惟一。如果和 Ant 插件的 id 相同，原 Ant 组就会被 myplugin2 视图组抹掉了（右图）。如果删除掉<category>不设置，则 Eclipse 会自动新增一个“其他”组，并将两视图加入（中图）。
- <view>的 category 是表示本视图属于哪个组，与上面<category>项的 id 值相同。
- <view>的 class 是视图所对应的类（还没编写，下一步将完成这两个类）。
- <view>的 id 是视图标识，建议设置成和 class 一样的名称。

图 18.4　显示视图窗口

18.2.2　创建视图类

在 18.2.1 小节的 plugin.xml 中提前设置了视图对应的类：cn.com.chengang.View1、

View2，本小节就来在包 cn.com.chengang 中创建这两个视图类。

视图的类必须继承抽象类 ViewPart，此类有两种方法 createPartControl、setFocus。我们要在 createPartControl 方法创建两个视图的界面组件，第一个视图创建一个列表，第二个视图创建一个文本框。而 setFocus 是关于视图焦点的方法，一般都不用写，让它空实现。

两类的代码如下：

```
//------文件名：View1.java----------
public class View1 extends ViewPart {
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        List list = new List(topComp, SWT.BORDER);
        list.add("中国");
        list.add("美国");
        list.add("法国");
    }
    public void setFocus() {}
}
```

```
//------文件名：View2.java----------
public class View2 extends ViewPart {
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("我是 text 框");
    }
    public void setFocus() {}
}
```

18.2.3　修改透视图类 SamplePerspective

在 18.1 节加入一个透视图时，建立了一个透视图类 SamplePerspective，当时有一个继承自接口的方法 createInitialLayout，它还是空实现的。本例将通过修改这种方法，将两个视图加入到透视图中。createInitialLayout 方法的代码如下：

```
//参数 IPageLayout 是用于透视图的布局
public void createInitialLayout(IPageLayout layout) {
    //得本透视图的编辑空间标识
    String editorArea = layout.getEditorArea();
    /*
     * 将视图 1 加入到透视图的左部
     * "left" 视图区的 id 标识为"left"
     * IPageLayout.LEFT 在透视图布局中的位置靠左
```

```
        * 0.2f 占用透视图 20%的宽度
        * editorArea 使用透视图的编辑空间
        */
        IFolderLayout left = layout.createFolder("left", IPageLayout.LEFT, 0.2f, editorArea);
        left.addView("cn.com.chengang.View1"); //参数为 plugin.xml 中视图 1 的 id 标识
        //将视图 2 加入到透视图的底部
        IFolderLayout bottom = layout.createFolder("bottom", IPageLayout.BOTTOM, 0.8f,
editorArea);
        bottom.addView("cn.com.chengang.View2");
        /*
        * 将 17.2.2 小节第 4 点所定义的 actionset（主菜单、工具栏按钮）加入到本透视图。这个效果要在
        * plugin.xml 文件的 action 设置中将 visible="false"才看得出效果，这时打
        * 开其他透视图，action 设置的主菜单、工具栏按钮将不会出现在界面上，只有打
        * 开本透视图才会出现，因为本透视图用下面的语句手工加入了此 action
        * 参数 myplugin2.actionSet 为 action 在 plugin.xml 文件中设置的 id 标识
        */
        layout.addActionSet("myplugin2.actionSet");
    }
```

18.2.4　运行插件

运行插件后，打开 myplugin 透视图，效果如图 18.5 所示。如果两个视图还没有显示在透视图上，则需要把透视图先关闭，再打开，以应用新的透视图设置。

图 18.5　加入两视图后的透视图

18.3　在视图之间实现事件监听

两个视图中的组件之间的互动，在开发插件时是经常碰到的问题。例如，在 18.2.4 小节的界面中，单击视图 1 列表中的某项时，视图 2 的文本框也作相应显示。本节将实现此功能。

18.3.1 修改 View1.java、View2.java

首先，要在两视图中互动就必须先解决“如何在视图 1 中取得视图 2 的对象”的问题。Eclipse 通过 plugin.xml 来加载插件和插件中的扩展点（如视图），所以可以由视图的 id 标识来取得视图对象，具体语句如下：

```
IWorkbenchPage wbp = getViewSite().getPage();
IViewPart view2 = wbp.findView("cn.com.chengang.View2");
```

得到了视图 2 的对象后，其他一切就都好办了，先给出修改后 View1 如下：

```
public class View1 extends ViewPart {
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        final List list = new List(topComp, SWT.BORDER);
        list.add("中国");
        list.add("美国");
        list.add("法国");
        //列表选择事件监听
        list.addSelectionListener(new SelectionListener() {
            public void widgetSelected(SelectionEvent e) {
                //由 IWorkbenchPage 取出 view2
                IWorkbenchPage wbp = getViewSite().getPage();
                IViewPart view2 = wbp.findView("cn.com.chengang.View2");
                //将当前选择的列表项显示在文本框中
                Text text = ((View2) view2).getText();
                text.setText(list.getSelection()[0]);
            }
            public void widgetDefaultSelected(SelectionEvent e) {}
        });
    }
    public void setFocus() {}
}
```

然后将 View2 的文本框对象改成类的实例变量，并编写它相应的 set/get 方法，代码如下：

```
public class View2 extends ViewPart {
    private Text text;
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        text = new Text(topComp, SWT.BORDER);
        text.setText("我是 text 框");
```

```
    }
    public void setFocus() {}
    //文本框 text 相应的 set/get 方法
    public Text getText() {return text;}
    public void setText(Text text) {this.text = text;}
}
```

18.3.2　总结

（1）在插件中 IWorkbenchPage 对象比较重要，在这里再给出一种获得此对象的方法。当不是在视图里而是在 Action 里要取 IWorkbenchPage 对象时，就可以用下面的方法：

```
Myplugin2Plugin.getDefault().getWorkbench().getActiveWorkbenchWindow().
getActivePage();
```

（2）IWorkbenchPage.findView("cn.com.chengang.View2")，中的参数为“视图 2”在 plugin.xml 中设置的 id 标识，由此可见 plugin.xml 文件在插件中的地位是极其重要的。IWorkbenchPage 除了 findView 方法之外，还有 findEditor 方法是用来得到编辑器的。

像“cn.com.chengang.View2”这种标识符在系统开发中会经常用到，最好建立一个常量专用类来集中放置这些字符串常量，然后系统中用的时候只用其常量名就行了，否则把标识符的字串分散在代码中，以后改起来会让你痛不欲生。

常量类的示意代码如下：

```
public final class StringConstants {
    public final stafic String VIEW1 = "cn.com.chengang.View1";
    public final static String VIEW2 = "cn.com.chengang.View2";
}
```

要用的时候则这样写：findView(StringConstants.VIEW2);

18.4　给视图加下拉菜单和按钮

本例将给视图加入下拉菜单和按钮，如图 18.6 所示。同时再为列表添加一个右键菜单，这样读者可以用来和视图的下拉菜单进行比较阅读。

图 18.6　效果图

18.4.1　创建 ActionGroup 类

加入菜单和按钮的方法与 SWT/JFace 组件的一样。先创建一个 ActionGroup 代码如下：

```
//--------文件名：MyActionGroup.java----------
public class MyActionGroup extends ActionGroup {
    /*
     * 加入按钮
     */
    public void fillActionBars(IActionBars actionBars) {
        if (actionBars == null)
            return;
        IToolBarManager toolBar = actionBars.getToolBarManager();
        toolBar.add(new Action1());
        toolBar.add(new Action2());
    }

    /*
     * 加入下拉菜单、右键弹出菜单
     */
    public void fillContextMenu(IMenuManager menu) {
        if (menu == null)
            return;
        menu.add(new Action1());
        menu.add(new Action2());
    }

    private class Action1 extends Action {
        public Action1() {
            ImageDescriptor imageDesc=WorkbenchImages.getImageDescriptor(IworkbenchGraphic
Constants.IMG_ETOOL_HOME_NAV);
            setHoverImageDescriptor(imageDesc);
            setText("Action1");
        }
        public void run() {}
    }

    private class Action2 extends Action {
        public Action2() {
            ImageDescriptor imageDesc = WorkbenchImages.getImageDescriptor(Iworkbench
GraphicConstants.IMG_ETOOL_IMPORT_WIZ);
            setHoverImageDescriptor(imageDesc);
            setText("Action2");
        }
        public void run() {}
```

```
    }
}
```

程序说明：

- 本程序中含有两个 Action 类：Action1、Action2，和以往的 Action 不同之处在于它的图像描述符是直接从 Eclipse 环境中取得。既然插件在 Eclipse 环境内运行，那么 Eclipse 环境本身的图标就可以直接拿来使用。
- fillContextMenu 方法比起以前的少了几句。在 18.4.2 小节，可以看到它移出到 View1 类中去了，主要原因是为了此方法兼顾添加视图的下拉菜单。

18.4.2　修改 View1 类

在 View1 中增加了三种方法，分别用来加入视图的导航栏按钮、下拉菜单，以及加入列表 List 的右键菜单。代码如下：

```
public class View1 extends ViewPart {
    private List list; //将 List 写成类的实例变量，以扩大它的可访问范围

    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        list = new List(topComp, SWT.BORDER);
        list.add("中国");
        list.add("美国");
        list.add("法国");
        //列表选择事件监听(和以前一样，省略)
        /*
         * 加入导航栏按钮、下拉菜单、右键菜单
         */
        MyActionGroup actionGroup = new MyActionGroup();
        fillViewAction(actionGroup);//加入视图的导航栏按钮
        fillViewMenu(actionGroup);//加入视图的下拉菜单
        fillListMenu(actionGroup);// 加入视图的下拉菜单
    }

    /**
     * 加入视图的导航栏按钮
     */
    private void fillViewAction(MyActionGroup actionGroup) {
        IActionBars bars = getViewSite().getActionBars();
        actionGroup.fillActionBars(bars);
    }

    /**
     * 加入视图的下拉菜单
     */
```

```
    private void fillViewMenu(MyActionGroup actionGroup) {
        IMenuManager menu = getViewSite().getActionBars().getMenuManager();
        actionGroup.fillContextMenu(menu);
    }

    /**
     * 加入列表 List 的右键菜单
     */
    private void fillListMenu(MyActionGroup actionGroup) {
        MenuManager menu1 = new MenuManager();
        Menu m = menu1.createContextMenu(list);
        list.setMenu(m);
        actionGroup.fillContextMenu(menu1);
    }

    public void setFocus() {}
}
```

程序说明：

- 过去写在 ActionGroup 中的两句移到了 fillListMenu 方法中。
- 视图加按钮、菜单的方式和以前 SWT/JFace 的方式是一样的。只不过以前用自己生成 MenuManager 对象等，而现在的插件就只需要使用视图已有的 MenuManager 对象。

18.5　加入编辑器（editors）

本节将给出如下的实例：双击视图 1 中的列表项，将在透视图中加入相应的编辑器。这种效果就像在 Eclipse 中双击 Java 源文件，就会打开该源文件相对应的编辑器一样。效果如图 18.7 所示。

图 18.7　编辑器效果图

和以前一样，先来修改 plugin.xml 文件将编辑器的扩展点加入，然后再创建相应的编辑

器类，最后编写列表双击的事件代码。

18.5.1　修改 plugin.xml 文件，设置三个编辑器的扩展点

```
<extension
      point="org.eclipse.ui.editors">
  <editor
        name="中国 Editor"
        icon="icons/project.gif"
        class="cn.com.chengang.ChinaEditor"
        id="cn.com.chengang.ChinaEditor">
  </editor>
  <editor
        name="美国 Editor"
        icon="icons/prev.gif"
        class="cn.com.chengang.UsaEditor"
        id="cn.com.chengang.UsaEditor">
  </editor>
  <editor
        name="法国 Editor"
        icon="icons/remove.gif"
        class="cn.com.chengang.FranceEditor"
        id="cn.com.chengang.FranceEditor">
  </editor>
</extension>
```

代码说明：

编辑器的扩展点是 org.eclipse.ui.editors，它各项的含义和视图扩展点基本一样，请参照 18.2.1 小节的视图扩展点的说明。这里强调一点：icon 也是必填项。

18.5.2　创建三个编辑器类

在 18.5.1 小节的 plugin.xml 中，提前设置了编辑器对应的类 cn.com.chengang.ChinaEditor 和 UsaEditor、FranceEditor，本小节就来在包 cn.com.chengang 中创建这三个编辑器类。

编辑器必须实现 IEditorPart 接口，但通常是继承抽象类 EditorPart 类（EditorPart 是 IEditorPart 的子类）。如果继承 EditorPart 则必须实现该抽象类的七种方法，在此先实现方法 init、createPartControl。本例只给出了 ChinaEditor 的代码，UsaEditor、FranceEditor 与之完全类似，ChinaEditor 的代码如下：

```
//------- 文件名：ChinaEditor.java ----------
public class ChinaEditor extends EditorPart {
    /**
     * Editor 的初始化方法，本方法前两句是固定不变的
     */
```

```
    public void init(IEditorSite site, IEditorInput input) throws PartInitException {
        System.out.println("init");
        setSite(site);
        setInput(input);
        //设置 Editor 标题栏的显示名称，不要，则名称用 plugin.xml 中的 name 属性
        //setPartName(input.getName());
        //设置 Editor 标题栏的图标，不要，则会自动使用一个默认的图标
        //setTitleImage(input.getImageDescriptor().createImage());
    }

    /**
     * 在此方法中创建 Editor 中的界面组件
     */
    public void createPartControl(Composite parent) {
        System.out.println("createPartControl");
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("中国之编辑器");
    }
    //此五个抽象类的方法下面将讲解，现在让它空实现
    public void doSave(IProgressMonitor monitor) {}
    public boolean isSaveAsAllowed() {return false;}
    public void doSaveAs() {}
    public boolean isDirty() {return false;}
    public void setFocus() {}
}
```

18.5.3　创建 IEditorInput

获取视图对象是用 IWorkbenchPage 的 findView 方法，方法参数是视图在 plugin.xml 中的 id 标识。获取编辑器对象是用 findEditor 方法，但该方法的参数却不是 id 标识的字符串，而是一个 IEditorInput 对象。另外，加载一个编辑器是用 IWorkbenchPage 的 openEditor (editorInput, editorID)方法。

由上可知，编辑器都会要对应一个 IEditorInput 和一个 EditorPart，而且在 IWorkbenchPage 中是根据 IEditorInput 来取得 EditorPart，如图 18.8 所示。

图 18.8　编辑器的组成

在本小节将要创建三个 Editor 相对应的 IEditorInput。在这里只给出了 ChinaEditor 对应

的 IEditorInput，其他两个与之类似，请读者查阅配书光盘的代码。

```
//--------- 文件名：ChinaEditorInput.java ----------
public class ChinaEditorInput implements IEditorInput {
    /**
     * 返回 true，则打开该编辑器后它会出现在 Eclipse 主菜单“文件”
     * 最下部的最近打开的文档栏中。返回 false 则不出现在其中
     */
    public boolean exists() {
        return true;
    }

    /**
     * 编辑器标题栏的图标，不过它还需要在 ChinaEditor 中用
     * setTitleImage 方法设置，才能出现在标题栏中
     */
    public ImageDescriptor getImageDescriptor() {
        return WorkbenchImages.getImageDescriptor(IWorkbenchGraphicConstants.IMG_
ETOOL_HOME_NAV);
    }

    /**
     * 编辑器标题栏的显示名称，和上面的 getImageDescriptor
     * 一样也要在 ChinaEditor 中用 setPartName 方法设置，才能出现在标题栏中
     */
    public String getName() {
        return "中国的编辑器";
    }

    /**
     * 编辑器标题栏的小黄条提示文字，不需像 getName 那样在 ChinaEditor 中再设置
     */
    public String getToolTipText() {
        return "这是视图 1 列表中的中国项对应的编辑器";
    }

    /**
     * 返回一个可以用做保存本编辑输入数据状态的对象，本例让它空实现
     */
    public IPersistableElement getPersistable() {
        return null;
    }

    /**
     * 得到一个编辑器的适配器，本例让它空实现
     *      IAdaptable a = new ChinaEditorInput();
```

```
     *      IFoo x = (IFoo)a.getAdapter(IFoo.class);
     *      if (x != null)
     *          [用 x 来做 IFoo 的事情]
     */
    public Object getAdapter(Class adapter) {
        return null;
    }
}
```

18.5.4　打开编辑器

有了 EditorPart 和 IEditorInput 后，就可以在 Eclipse 中打开编辑器了。本例是要实现双击视图 1 的列表，则会打开列表项相对应的编辑器，因此在 View1 类的 List 对象添加一个鼠标双击事件监听器。另外还要考虑到，如果已经打开了列表项对应的编辑器，则下次再双击时就不再打开该项的编辑器，而是将其设成当前选择的编辑器。

查找编辑器的方法是：IEditorPart editor = IWorkbenchPage.findEditor(IEditorInput);

打开编辑器的方法是：IWorkbenchPage.openEditor(IEditorInput, editorID);

所有代码如下：

```
list.addMouseListener(new MouseAdapter() {
    ChinaEditorInput chinaEditorInput = new ChinaEditorInput();
    UsaEditorInput usaEditorInput = new UsaEditorInput();
    FranceEditorInput franceEditorInput = new FranceEditorInput();

    public void mouseDoubleClick(MouseEvent e) {
        /*
         * 根据列表不同项得到其相应的 editorInput 和 editorID，其中
         * editorID 指该编辑器在 plugin.xml 文件中设置 id 标识值
         */
        List list = (List) e.getSource();//由 MouseEvent 得到列表对象
        String listStr = list.getSelection()[0];//得到列表当前项字符
        IEditorInput editorInput = null;
        String editorID = null;
        if (listStr.equals("中国")) {
            editorInput = chinaEditorInput;
            editorID = "cn.com.chengang.ChinaEditor";
        } else if (listStr.equals("美国")) {
            editorInput = usaEditorInput;
            editorID = "cn.com.chengang.UsaEditor";
        } else if (listStr.equals("法国")) {
            editorInput = franceEditorInput;
            editorID = "cn.com.chengang.FranceEditor";
        }
        //如果 editorInput 或 editorID 为空则中断返回
```

```
            if (editorInput == null || editorID == null)
                return;
            //取得 IWorkbenchPage，并搜索使用 editorInput 对象对应的编辑器
            IWorkbenchPage workbenchPage = getViewSite().getPage();
            IEditorPart editor = workbenchPage.findEditor(editorInput);
            /*
             * 如果此编辑器已经存在，则将它设为当前的编辑器（最顶端），否则
             * 重新打开一个编辑器
             */
            if (editor != null) {
                workbenchPage.bringToTop(editor);
            } else {
                try {
                    workbenchPage.openEditor(editorInput, editorID);
                } catch (PartInitException e2) {
                    e2.printStackTrace();
                }
            }
        }
});
```

程序说明：

在本程序中为了便于理解，使用了 if…else 这种简单的方式来判断双击的列表项，这适合列表项较少的情况，如果列表项太多了，则代码就会相当长。解决这个问题，可将各 IEditorInput 对象与列表 List 的项对应起来，并将 eidtorID 写到各 IEditorInput 类中。这样就可以用下面的方式来得到 IEditorInput 和 eidtorID 了。

```
String key = "" + list.getSelectionIndex();
IEditorInput editorInput = (IEditorInput) list.getData(key);
String eidtorID = editorInput.getEditorID();
```

18.5.5　总结

在实际开发中很多界面都是创建在编辑器上，虽然在这里只讲了最常用的编辑器使用方法，但以足够应付大部分开发的需要。如果你想了解更多关于编辑器的信息，可以查阅编辑器的帮助文档，它在帮助中的位置是“平台插件开发者指南→程序员指南→编辑器”。

18.6　编辑器类（EditorPart）方法使用说明

在 18.5 节将 ChinaEditor 类继承 EditorPart 抽象类时，只实现了两种方法：init、createPartControl，本节将通过“EditorPart 方法的执行情况”、“各方法的作用及含义”、“一个实例”来逐步讲解其他的五种方法。

18.6.1　EditorPart 方法的执行情况

要使用好 EditorPart，首先得了解其方法在各种情况下的执行流程，我们在类的每一种方法前加上 System.out.println（"方法名：***"），运行后就可以得到如下的结果。

（1）双击列表项打开编辑器时：init→isDirty→createPartControl→isDirty→isDirty→isDirty→isDirty→isDirty→setFocus→isDirty→isSaveAsAllowed。

（2）关闭编辑器时：setFocus→isDirty→isSaveAsAllowed→isDirty→isSaveAsAllowed→setFocus→isDirty，如果保存编辑器，则最后还会执行 doSave 方法。

（3）单击编辑器标题时：setFocus。

（4）编辑器失去焦点时：isDirty→isSaveAsAllowed→isDirty→isSaveAsAllowed。

（5）编辑器得到焦点时：setFocus→isDirty→isSaveAsAllowed→isDirty→isSaveAs Allowed。

（6）当编辑器可以保存，并在主菜单“文件→保存”选项或按 Ctrl+S 键时：isDirty→doSave。

18.6.2　各方法的作用及含义

1．boolean isDirty()

由此方法获知编辑器是否脏了（所谓“脏”是指编辑器中的值已经发生了改变），true 表示脏。当其返回 true 时，会出现两个效果：编辑器的标题前出现一个“*”号，主菜单“文件”下的“保存”项可用。

特别要注意的是，编辑器不会自己判断是否脏了，这需要在程序中用语句手动设置，例如，在编辑器的文本框加入一个键盘监听事件，当在文本框中输入字符时，则将 isDirty 方法返回值设为 true（脏）。

在方法执行情况中，可以看到此方法的执行是最频繁的，所以不要在这种方法中加入太多的执行语句，否则会影响程序执行速度。

2．void doSave ()

在这种方法中编写保存编辑器的代码，当在主菜单中选择“文件→保存”选项时会执行此方法，但在 isDirty 返回 true 时“保存”菜单和 Ctrl+S 键才可以用，也即 isDirty 方法控制着 doSave 方法的执行。

当保存成功时，要注意将脏的状态设回 false，并调用 firePropertyChange 方法将编辑器的界面状态更新（编辑器标题前的“*”号及“保存”菜单）。

3．boolean isSaveAsAllowed()

是否允许编辑器使用“另存为”功能。如果此项返回 false，则不能使用“另存为”功能，而且主菜单“文件”下的“另存为”项被置灰。

4．void doSaveAs()

和 doSave 的作用相似，在这里书写“另存为”功能的处理代码。

5. void setFocus()

当编辑器获得焦点时执行该方法。

18.6.3　一个实例

在这个实例中，当修改 ChinaEditor 编辑器中文本框的文字时，编辑器标题前出现“*”且主菜单“文件”下的“保存”选项可用。当保存了编辑器后，“*”消失并且“保存”菜单不可用。当编辑器为脏时，关闭编辑器会弹出一个提示框（如图 18.9 所示）。

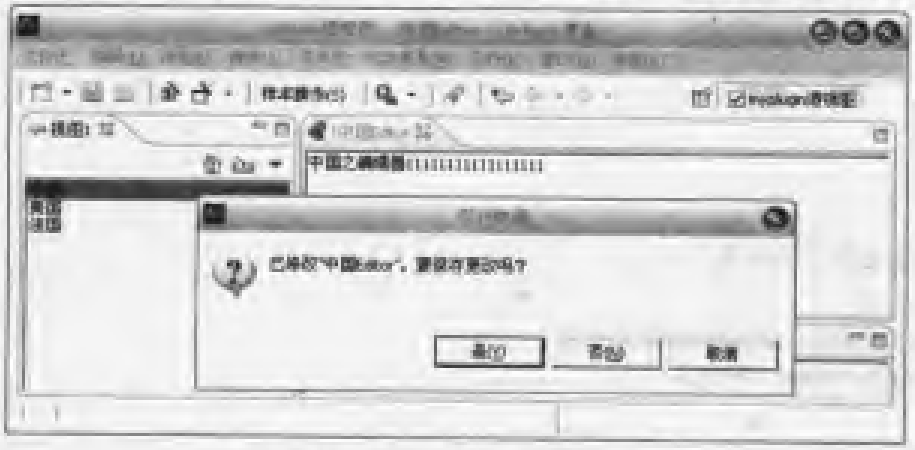


图 18.9　关闭脏编辑器时的效果图

要实现以上效果只需要修改 ChinaEditor 类，修改后的代码如下：

```
public class ChinaEditor extends EditorPart {
    private boolean dirty = true; //编辑器是否为脏的标识

    //……init 方法不变，省略

    /**
     * 在此方法中创建 Editor 中的界面组件
     */
    public void createPartControl(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new FillLayout());
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("中国之编辑器");
        text.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e) {
                //如果编辑器不脏（即没有修改），则标志它脏并刷新界面状态
                if (!isDirty()) {
                    setDirty(true);
                    firePropertyChange(IEditorPart.PROP_DIRTY);
                }
            }
```

```
        });
    }

    /**
     * 保存的处理代码在这种方法中，当按 Ctrl+S 键时会执行此方法。
     * 最后别忘记标志为非脏及刷新界面状态
     */
    public void doSave(IProgressMonitor monitor) {
        if (isDirty()) {
            //……保存编辑器事件处理代码（省略）
            setDirty(false);
            firePropertyChange(IEditorPart.PROP_DIRTY);
        }
    }

    /**
     * 是否允许“另存为”
     */
    public boolean isSaveAsAllowed() {
        return false; //不允许
    }

    /**
     * “另存为”的代码写在这里，本例不实现它
     */
    public void doSaveAs() {}

    /**
     * dirty 标识的 set 方法，由此方法设置编辑器为脏
     */
    public void setDirty(boolean dirty) {
        this.dirty = dirty;
    }

    /**
     * 编辑器的内容是否脏了，true 脏,false 不脏
     */
    public boolean isDirty() {
        return dirty;
    }

    /**
     * 当编辑器获得焦点时会执行此方法，本例空实现
     */
    public void setFocus() {}
}
```

程序说明：

firePropertyChange(IEditorPart.PROP_DIRTY);这一句除了能将界面状态刷新之外，如果 IEditorPart 添加了如下的监听器，则还可以触发其中的 propertyChanged 事件。

```
chinaEditor.addPropertyListener(new IPropertyListener() {
    //此时 source 为 ChinaEditor 对象，propId 为 IEditorPart.PROP_DIRTY 这个常量值
    public void propertyChanged(Object source, int propId) {
        //事件处理代码，省略
    }
});
```

18.7　加入首选项（preferencePages）

选择主菜单“窗口→首选项”选项打开“首选项”窗口，如图 18.10 所示。这个窗口是 Eclipse 所有设置项的集中地，同样也常是第三方插件进行设置的窗口。图中左边两个方框标注的就是 SWT Designer 插件的设置树和本书插件 MyPlugin 的设置树。

本节将实现 MyPlugin 的首选项中的设置树。

图 18.10　“首选项”窗口

18.7.1　修改 plugin.xml 文件，设置首选项的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前：

```
<extension point="org.eclipse.ui.preferencePages">
    <page
      name="myplugin2 插件设置"
      class="cn.com.chengang.preferences.RootPreferencePage"
      id="cn.com.chengang.preferences.RootPreferencePage">
    </page>
    <page
```

```
        name="DB 数据库"
        category="cn.com.chengang.preferences.RootPreferencePage"
        class="cn.com.chengang.preferences.DBPreferencePage"
        id="cn.com.chengang.preferences.DBPreferencePage">
      </page>
</extension>
```

代码说明：

- org.eclipse.ui.preferencePages 是首选项的扩展点。
- name 是首选项的树结点的名称。
- class 是首选项的树结点所对应的类（还没编写，下一步将完成此类）。
- id 是首选项的树结点标识，建议设置成和 class 一样的名称。
- category 是要等于父结点的 id 标识。

18.7.2　建立首选项各结点对应的类

在 18.7.1 小节的 plugin.xml 中提前设置了首选项结点对应的类 RootPreferencePage、DBPreferencePage，本小节就来在包 cn.com.chengang.preferences 中创建此类。

首选项的类必须继承 PreferencePage 抽象类和实现 IWorkbenchPreferencePage 接口，实现接口只有一种方法 init，抽象类则有一些“首选项”窗口的按钮的执行方法。本小节实例先给出代码简单一些的根结点 RootPreferencePage 类，再给出复杂一些的子结点 DBPreferencePage 类，两类具体代码如下：

```
//--------文件名：RootPreferencePage.java----------------------
public class RootPreferencePage extends PreferencePage
                                    implements IWorkbenchPreferencePage {

    public void init(IWorkbench workbench) {}

    protected Control createContents(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new RowLayout());
        new Label(topComp, SWT.NONE).setText("欢迎使用 myplugin2 插件");
        return topComp;
    }
}
```

```
//--------文件名：DBPreferencePage.java----------------------
public class DBPreferencePage extends PreferencePage
                        implements IWorkbenchPreferencePage, ModifyListener {

    //为文本框定义三个键值
    public static final String URL_KEY = "$URL_KEY";
```

```
public static final String USERNAME_KEY = "$USERNAME_KEY";
public static final String PASSWORD_KEY = "$PASSWORD_KEY";
//为文本框值定义三个默认值
public static final String URL_DEFAULT = "jdbc:db2://127.0.0.1/mydb";
public static final String USERNAME_DEFAULT = "glchengang";
public static final String PASSWORD_DEFAULT = "12345678";
//定义三个文本框
private Text urlText;
private Text usernameText;
private Text passwordText;
//定义一个 IPreferenceStore 对象
private IPreferenceStore ps;

/**
 * 接口 IWorkbenchPreferencePage 的方法，负责初始化。在此方法中设置一个
 * PreferenceStore 对象，由此对象提供文本框值的读入/写出方法
 */
public void init(IWorkbench workbench) {

setPreferenceStore(Myplugin2Plugin.getDefault().getPreferenceStore());
}

/**
 * 父类的界面创建方法
 */
protected Control createContents(Composite parent) {
    Composite topComp = new Composite(parent, SWT.NONE);
    topComp.setLayout(new GridLayout(2, false));
    /*
     * 创建三个文本框及其标签
     */
    new Label(topComp, SWT.NONE).setText("URL:");
    urlText = new Text(topComp, SWT.BORDER);
    urlText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

    new Label(topComp, SWT.NONE).setText("用户名:");
    usernameText = new Text(topComp, SWT.BORDER);
    usernameText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));

    new Label(topComp, SWT.NONE).setText("密码:");
    passwordText = new Text(topComp, SWT.BORDER | SWT.PASSWORD);
    passwordText.setLayoutData(new GridData(GridData.FILL_HORIZONTAL));
    //取得一个 IPreferenceStore 对象
    ps = getPreferenceStore();
    /*
```

```
        * 取出以前保存的值，并将其设置到文本框中，如果取出值为空
        * 或者是空字串，则填入默认值
        */
       String url = ps.getString(URL_KEY);
       if (url == null || url.trim().equals(""))
           urlText.setText(URL_DEFAULT);
       else
           urlText.setText(url);

       String username = ps.getString(USERNAME_KEY);
       if (username == null || username.trim().equals(""))
           usernameText.setText(USERNAME_DEFAULT);
       else
           usernameText.setText(username);

       String password = ps.getString(PASSWORD_KEY);
       if (password == null || password.trim().equals(""))
           passwordText.setText(PASSWORD_DEFAULT);
       else
           passwordText.setText(password);
       /*
        * 添加事件监听，this 代表本类，因本类也实现了 ModifyListener 接口，
        * 所以本类可以作为监听器使用
        */
       usernameText.addModifyListener(this);
       passwordText.addModifyListener(this);
       urlText.addModifyListener(this);
       return topComp;
   }

   /**
    * 本类实现 ModifyListener 接口的方法，当三个文本框中发生修改时将执行此方法。
    * 方法中对输入值进行了验证并将“确定”、“应用”两按钮使能
    */
   public void modifyText(ModifyEvent e) {
       String errorStr = null;//将原错误信息清空
       if (urlText.getText().trim().length() == 0) {
           errorStr = "URL 不能为空！";
       } else if (usernameText.getText().trim().length() == 0) {
           errorStr = "用户名不能为空！";
       } else if (passwordText.getText().trim().length() == 0) {
           errorStr = "密码不能为空！";
       }
       setErrorMessage(errorStr);//errorStr=null 时复原为正常的提示文字
       setValid(errorStr == null);//“确定”按钮
```

```
        getApplyButton().setEnabled(errorStr == null);// "应用" 按钮
    }

    /**
     * 父类方法，单击 "复原默认值" 按钮时将执行此方法，取出默认值设置到文本框中
     */
    protected void performDefaults() {
        urlText.setText(URL_DEFAULT);
        usernameText.setText(USERNAME_DEFAULT);
        passwordText.setText(PASSWORD_DEFAULT);
    }

    /**
     * 父类方法，单击 "应用" 按钮时执行此方法，将文本框值保存并弹出成功的提示信息
     */
    protected void performApply() {
        doSave(); //自定义方法，保存设置
        MessageDialog.openInformation(getShell(), "信息", "成功保存修改!");
    }

    /**
     * 父类方法，单击 "确定" 按钮时执行此方法，将文本框值保存并弹出成功的提示信息
     * @return true 成功退出
     */
    public boolean performOk() {
        doSave();
        MessageDialog.openInformation(getShell(),"信息","修改在下次启动生效");
        return true;
    }

    /**
     * 自定义方法，保存文本框的值
     */
    private void doSave() {
        ps.setValue(URL_KEY, urlText.getText());
        ps.setValue(USERNAME_KEY, usernameText.getText());
        ps.setValue(PASSWORD_KEY, passwordText.getText());
    }
}
```

18.7.3　运行插件

运行插件后，打开新 Eclipse 环境的 "首选项" 窗口，选择 "DB 数据库" 选项，将其中的密码删除后可得到如图 18.11 所示的出错效果。

图 18.11　首选项出错的效果图

18.7.4　总结

本例程的核心是 IPreferenceStore 对象的使用，用它的 getString 方法来取值、setValue 方法来存值。其次和以前的事件代码写法有所不同的是：本类实现了 ModifyListener 接口，也成为了一个监听器，这样在各文本框的加入监听器的代码就会简洁很多，不过其事件代码必须保证三个文本框可以共用才行。

18.8　加入帮助（toc）

如图 18.12 所示，本节将为 myplugin2 插件加入帮助。本节实例将演示如何在 plugin.xml 添加扩展点，如何创建帮助左部的结点树，如何链接到帮助文件。

图 18.12　帮助

18.8.1　修改 plugin.xml 文件，设置三个帮助的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前：

```
<extension point="org.eclipse.help.toc">
  <toc
        file="toc.xml"
        primary="true">
  </toc>
  <toc
        file="other_toc.xml"
        primary="true">
  </toc>
</extension>
```

说明：

- org.eclipse.help.toc 是帮助的扩展点。
- toc 是帮助目录项的设定。
- toc 的 file 是指定帮助目录文件（还没编写，18.8.2 小节将完成这两个 xml 文件）。
- toc 的 primary 是该项帮助目录是否显示在帮助窗口中，true 为显示。
- 如图 18.12 所示左边的结点树，toc.xml 对应于“myplugin2 帮助”结点，other_toc.xml 对应于“myplugin2 的新版功能”结点。

18.8.2　编写帮助目录文件 toc

这两个文件应放置在项目的根目录下，和 src 目录平级，如图 18.13 所示。

图 18.13　目录结构图

先给出第一个文件 toc.xml 的代码，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<toc label="myplugin2 帮助">
```

```
    <topic label="透视图" href="contexts/1.html">
        <topic label="透视图简介" href="contexts/1_1.html"/>
        <topic label="视图" href="contexts/1_2.html">
            <topic label="视图－创建" href="contexts/1_2_1.html"/>
            <topic label="视图－使用" href="contexts/1_2_2.html"/>
            <topic label="视图－实例" href="contexts/1_2_3.html"/>
        </topic>
        <topic label="编辑器" href="contexts/1_3.html"/>
    </topic>
    <topic label="首选项" href="contexts/2.html"/>
    <topic label="控制台" href="contexts/3.html"/>
    <topic label="插件网站" href="http://blog.csdn.net/gichengang"/>
</toc>
```

说明：

- topic：设置帮助目录中的结点。
- topic 的 lable：结点显示的名称。
- topic 的 href：结点对应的帮助文件的路径及文件名，也可以是网址（还没编写对应的帮助文档，18.8.1 小节将完成它们）。
- 这里应注意一个细微的地方：xml 中项的结束符有两种，如

  <topic label="首选项" href="contexts/2.html"/>

  或者是：

  <topic label="首选项" href="contexts/2.html">< /topic>

  这两者的效果是一样的。

下面再给出另一个帮助目录文件 other_toc.xml，代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<toc label="myplugin2 的新版功能">
    <topic label='自制绘图" href="contexts/other_toc.html"/>
</toc>
```

18.8.3　创建相应的帮助文档

帮助文件创建在插件项目的根目录下，和 src 目录平级，目录结构如图 18.13 所示。帮助文件都是一些标准的 html 格式的网页文件，这里仅给出<topic label="透视图" href="contexts/1.html">项对应的 1.html 文件，其内容如下：

```
<HTML>
    <BODY>
        透视图 <A HREF="other/by_1_link.html">一个链接</A>
    </BODY>
</HTML>
```

在 1.html 中，有一个链接“other/by_1_link.html”，这个文件并不需要在帮助目录文件

toc.xml 中注册。

18.8.4　总结

创建帮助一般都分三步走：

（1）在 plugin.xml 添加扩展点。

（2）编写帮助目录的 toc 文件。

（3）编写相应的帮助文档。

还有一种做法是将帮助单独写成一个插件，这种插件和 myplugin2 插件没有什么不同，其开发过程也和本节一样，只不过该插件里只含有帮助的文件。这种方式适合比较大的项目使用，因为分成了没有什么联系的两个插件，这样就可以让一批人开发 myplugin2 插件，另一批人开发帮助插件，两队人马互不干扰。

18.9　弹出信息式的帮助（contexts）

弹出信息可以和窗口组件关联在一起，给用户显示有针对性的帮助，其效果如图 18.14 所示。如果组件设置有弹出信息，按 F1 键就可以激活它。

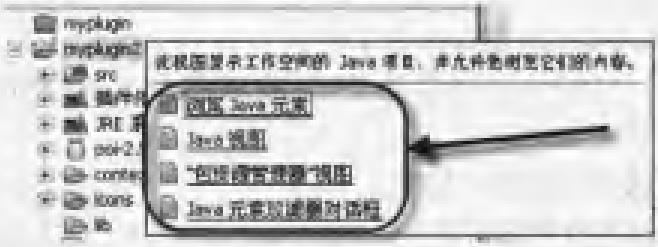


图 18.14　弹出信息

下面就以一个实例来演示如何实现弹出信息。

18.9.1　修改 plugin.xml 文件，设置弹出信息的扩展点

打开 plugin.xml 文件的编辑框，将如下代码块插入到最后一行的</plugin>项之前。

```
<extension point="org.eclipse.help.contexts">
    <contexts
            file="HelpContexts.xml">
    </contexts>
</extension>
```

代码说明：

- org.eclipse.help.contexts 是弹出信息的扩展点。
- contexts 的 file 指定弹出信息的设置文件（尚未编写，下一步将完成此 xml 文件）。

18.9.2　编写弹出信息的设置文件 HelpContexts.xml

HelpContexts.xml 文件应放置在项目的根目录下，和 src 目录平级。

```
<?xml version="1.0" encoding="UTF-8"?>
<contexts>
    <context id="textHelpId">
        <description>编辑器</description>
        <topic href="contexts/1_3.html"  label="编辑器的帮助"/>
    </context>
    <context id="buttonHelpId">
        <description>视图</description>
        <topic href="contexts/1_2_1.html"  label="视图－创建"/>
        <topic href="contexts/1_2_2.html"  label="视图－使用"/>
        <topic href="contexts/1_2_3.html"  label="视图－实例"/>
    </context>
</contexts>
```

代码说明：

- 这里设置了两个弹出信息 textHelpId、buttonHelpId。
- id：弹出信息的标识。
- description：弹出信息时的标题栏文字。
- href：弹出信息子项所对应的帮助文件，可以和 18.8 节的帮助文件共用，也可以自己设定单独的帮助文件。
- label：弹出信息子项的显示名称。

HelpContexts.xml 设置的效果如图 18.15 所示。

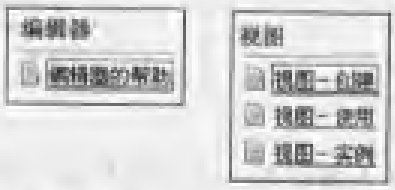


图 18.15　HelpContexts.xml 设置的效果图

18.9.3　创建弹出信息对应的帮助文件

由于在 HelpContexts.xml 中，href 项设置的都是 18.8 节的帮助文件，所以本例此步省略。

18.9.4　在界面组件中设置弹出信息

在界面组件中设置弹出信息分成以下三步。

（1）弹出信息的准备工作已经完成，接下来是将它们和对应组件关联起来。方法如下：

```
WorkbenchHelp.setHelp(text, "myplugin2.textHelpId");
```

第一个参数是组件对象名，第二个参数是弹出信息的 id 标识，前面要加上 plugin.xml 的<plugin>项设置的 id 标识。

（2）创建一个自定义 Dialog 来演示弹出信息。代码如下：

```
//--------- 文件名：HelpContextsDialog.java ----------
public class HelpContextsDialog extends Dialog {
    protected HelpContextsDialog(Shell parentShell) {
        super(parentShell);
    }

    protected Control createDialogArea(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new RowLayout());
        //界面组件
        Text text = new Text(topComp, SWT.BORDER);
        text.setText("编辑器                    ");
        Button button = new Button(topComp, SWT.BORDER);
        button.setText("  打开视图  ");
        //让弹出信息和界面组件关联起来
        WorkbenchHelp.setHelp(text, "myplugin2.textHelpId");
        WorkbenchHelp.setHelp(button, "myplugin2.buttonHelpId");
        return topComp;
    }
}
```

（3）编写打开此 Dialog 的方法，本例选择视图 1 上的一个按钮来打开 Dialog。在 MyActionGroup 类的 Action1 中的 run 方法中加入如下语句：

```
HelpContextsDialog dialog = new HelpContextsDialog(null);
dialog.open();
```

18.9.5　运行插件

运行插件后的效果如图 18.16 所示。

图 18.16　实例的弹出信息界面

18.9.6　总结

设置弹出信息和设置帮助（toc）有其类似之处，而且它们的 HTML 格式的帮助文件也可以共用，但这并不是说弹出信息要依赖于帮助（toc）的设置。而且不是所有的组件都支持弹出信息。下面给出不支持弹出信息的组件例表：

- ToolItem；

- CtabItem；
- TabItem；
- TableColumn；
- TableItem；
- TableTreeItem；
- TreeItem。

18.10　本 章 小 结

本章介绍了插件中最常用的扩展点，不仅讲述了扩展点的含义，而且给出相应的实例和具体的实现步骤。本章的每节组合在一起就是一个插件开发的开发流程，从透视图开始到帮助结束，插件开发大致是按照这个流程来走。

第 19 章　Eclipse 插件的国际化

19.1　国际化简介

Eclipse 的语言包是最好的国际化例子。Eclipse 一般是先发布英文版，然后不久之后便会提供相应版本的国际化语言包，这个“语言包”含有包括中文在内的几种主要语言。如果软件要走向国际市场，国际化是必须要做的一件事。

本章将继续基于原有 myplugin2 项目的基础上，通过一个实例来演示如何实现软件的国际化。本章主要内容是讲国际化的具体操作方法，但在翻译时还有很多是要考虑的，例如，大陆的“项目”，台湾地区叫“专案”，一些数字、日期、时间、货币、电话号码、邮递地址、个人称谓、度量衡、尊称和个人头衔等，在各种语言文字中都有着差异。

19.2　为国际化创建一个插件的“段项目”

像 Eclipse 的语言包一样，国际化语言包本身也是一个插件类项目，因此第一步是为语言包创建一个“段项目”。

（1）选择主菜单“文件→新建→项目”选项。如下图 19.1 所示，在弹出窗口中选择“插件开发→段项目”选项，然后单击“下一步”按钮。

图 19.1　新建项目

（2）在如图 19.2 所示的窗口中输入项目名（可以任意取名），本文取名为 myplugin2.language。其他设置不变，然后单击“下一步”按钮。

图 19.2　输入项目名

（3）在图 19.3 所示的窗口中设置父插件为 myplugin2，插件版本为 1.0.0，然后单击“完成”按钮。也可以通过“浏览”按钮选择父插件。

（4）经过以上三步完成了“段项目”的建立，其项目结构如图 19.4 所示。从图中可见“段项目”是没有 plugin.xml 文件的，取而代之的是 fragment.xml 文件，项目中仅有一个名为 src 的源文件目录。

图 19.3　设置父插件

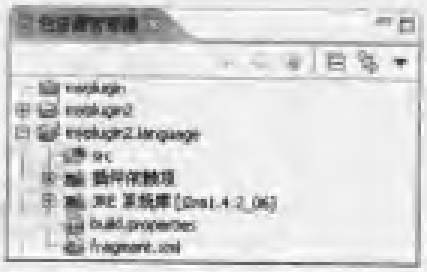


图 19.4　“段项目”的项目结构图

19.3　类程序的国际化

如图 19.5 所示，Eclipse 是这样进行国际化的：将 myplugin2 项目中需要国际化的字符

串抽取到 myplugin2.languaage 项目的一个文本文件中，此文本文件的扩展名是 properties，称之为资源文件，而 myplugin2 和 myplugin2.language 之间的映射关系则由 Message 类来负责。

图 19.5　国际化机制示意图

19.3.1　国际化前的准备工作

（1）为了不影响以前的程序代码，新创建一个自定义 Dialog，它包含两个按钮，本小节就对这两个按钮的文字进行国际化。

自定义 Dialog 的代码如下：

```
//------- 文件名：LanguageDialog.java --------
public class LanguageDialog extends Dialog {
    protected LanguageDialog(Shell parentShell) {
        super(parentShell);
    }

    protected Control createDialogArea(Composite parent) {
        Composite topComp = new Composite(parent, SWT.NONE);
        topComp.setLayout(new RowLayout());
        Button button1 = new Button(topComp, SWT.BORDER);
        button1.setText("确定");
        Button button2 = new Button(topComp, SWT.BORDER);
        button2.setText("删除");
        return topComp;
    }
}
```

（2）在视图 1 的 Action2 的 run 方法中，加入打开 Dialog 的语句。代码如下：

```
private class Action2 extends Action {
    public Action2() {……省略}
    public void run() {
        LanguageDialog dialog = new LanguageDialog(null);
        dialog.open();
    }
}
```

这样就可以通过图 19.6 中圆圈标识的工具栏按钮，弹出 LanguageDialog 对话框。

图 19.6　准备工作完成后的效果图

19.3.2　创建 Message 类

前面的准备工作完成后，这一步来创建最核心的 Message 类。Message 类的代码量很小，如下：

```
public class Messages {
    private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle
("cn.com.chengang.messages");

    public static String getString(String key) {
        try {
            return RESOURCE_BUNDLE.getString(key);
        } catch (MissingResourceException e) {
            return '!' + key + '!';
        }
    }
}
```

代码说明：

- ResourceBundle 是一个静态资源束对象，其字符中参数 cn.com.chengang.messages 中的 messages 是指 myplugin2.language 项目中资源文件的主名称，cn.com.chengang 是资源文件所在的包名。
- ResourceBundle 的 getString 方法的作用是从资源文件中取出与键值 key 对应的文字。

19.3.3　创建资源文件（*.properties）

资源文件包含了各种语言的翻译信息，它是一个扩展名为 properties 的文本文件。每一种语言文字对应一个资源文件，取名的方式是“主名+语言标识符+properties”。例如：

- messages.properties（默认，本例将默认的文档翻译成英文）；
- messages_en.properties（英文）；

- messages_zh_CN.properties（中文.大陆简体）；
- messages_zh_TW.properties（中文.台湾繁体）；
- messages_zh_HK.properties（中文.香港繁体）；
- messages_ja.properties（日文）等。

在 Eclipse 选择主菜单“窗口→首选项”选项，然后选择“插件开发→目标平台”选项，在“语言环境”一项的下拉框里可以找到更多语言标识符。

本例将为 LanguageDialog 的“确定”和“删除”按钮提供三种语言文字的翻译：英文、简体、繁体，如表 19.1 所示。

表 19.1　翻译对照表

key（键值）	英　文	简　体	繁　体
LanguageDialog.ok	OK	确定	確定
LanguageDialog.remove	Remove	删除	移除

给出各语言文件的具体内容如下：

文件一：messages.properties（把英文设为默认资源文件）。

```
LanguageDialog.ok=OK
LanguageDialog.remove=Remove
```

文件二：messages_zh_CN.properties（中文.大陆简体）。

```
LanguageDialog.ok=\u786E\u5B9A
LanguageDialog.remove=\u5220\u9664
```

文件三：messages_zh_TW.properties（中文.台湾繁体）。

```
LanguageDialog.ok=\u78BA\u5B9A
LanguageDialog.remove=\u79FB\u9664
```

在这里可以看到，简体和繁体的文字是由 UNICODE 码表示的，反斜杠加五个字符表示一个汉字。那么如何得到汉字相对应的 UNICODE 码呢？有如下三种方法：

（1）找个汉字与 UNICODE 码的映射表，把它背下来（此方法难度太大，不建议使用）。

（2）使用 JDK 的 bin 目录下的 native2ascii.exe 程序来转换（此方法简单易行，可以考虑使用）。以简体 messages_zh_CN.properties 的转换为例，先建立一个名为 messages_zh_CN.properties.src 的文件，文件中的翻译使用正常汉字：

```
LanguageDialog.ok=确定
LanguageDialog.remove=删除
```

然后用 native2ascii.exe 程序执行如下命令，就可以得到转成 UNICODE 码后的 messages_zh_CN.properties 文件。

```
native2ascii  messages_zh_CN.properties.src  messages_zh_CN.properties
```

（3）使用第三方的 Eclipse 的国际化插件（此方法简单易行且直观，推荐使用）。

有一个免费的国际化插件 jinto，下载网址是 http://www.guh-software.de/。该插件使用很简单，本书不再详述，下面给出一些使用界面截图，如图 19.7 至图 19.9 所示。

图 19.7　打开资源文件

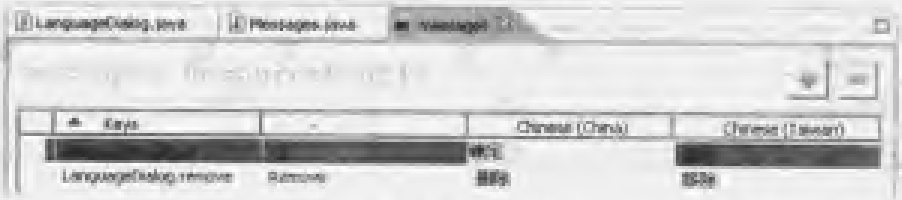


图 19.8　编辑资源文件

图 19.9　增加一种新语言的资源文件

19.3.4　修改 myplugin2 项目的程序

以上将 Message 类和资源文件都完成了，最后一步是修改 myplugin2 项目的程序。将原来的两句：

```
button1.setText("确定");
button2.setText("删除");
```

改为：

```
button1.setText(Messages.getString("LanguageDialog.ok"));
```

```
button2.setText(Messages.getString("LanguageDialog.remove"));
```

19.3.5　运行插件，测试国际化

在中文 Windows 下插件默认是以中文的环境运行，如果要测试国际化插件在其他语言环境下是否工作良好，就需要改变 Eclipse 的运行环境参数。

在主菜单选择“运行→运行(N)…”选项，显示出如图 19.10 所示的对话框。在“程序自变量环境”项中把-ln 参数的 zh_CN 改为 en 后，再运行插件，即可测试英文语言包的运行情况。同样，要测试繁体，将参数改为 zh_TW 即可。

如果设置的语言环境所对应的资源文件未找到，则会使用默认的 messages.properties。例如，将运行参数改为 ko（韩文），则 Eclipse 会先去查找 messages_ko.properties 文件，如果没有，就会转而查找 messages.properties。如果 messages.properties 存在，则即使内容并非韩文的翻译，Eclipse 也照样会使用它；如果 messages.properties 也不存在，则会抛出一个异常。

由上可知，messages.properties 是非常重要的，而且它的内容并不硬性地规定是英文或任何一种文字，将中文、德文放在其中也可以。但一般来说，还是建议 messages.properties 包含英文翻译。

图 19.10　运行环境设置

19.3.6　实践建议

本小节的实践建议如下：

- 建议一个包对应一个资源文件。不要整个系统都使用一个资源文件来翻译，这样单

个文档的体积就太大了，不利于维护；当然，也不必一个类对应一个资源文件，这样资源文件又太多了。

- 建议资源文件的路径和其翻译类的路径相同（两个插件保持目录结构相同），这样在今后查找类对应的资源文件时就会很方便。当然，Eclipse 也允许将资源文件放在其他路径，这时 Message 类中 ResourceBundle.getBundle("cn.com.chengang.messages") 的参数也要作相应改变。
- 建议将默认的 messages.properties 文件放在 myplugin2 项目对应的目录 cn.com.chengang 下，这样即使用户没有安装 myplugin2.language 插件，程序依然还有一个默认的资源文件使用。
- 建议 Message 类命名为 Message，或者用包名+Message。
- 建议资源文件的主名和 Message 类的名称相同，这样方便查找，否则要打开 Message 类查看 ResourceBundle.getBundle("cn.com.chengang.messages")的参数才知道它对应的是哪个文档。例如：Message 类名称为 Message，则文档为 message.properties、message_zh_CN.properties。
- 建议字符串项的键名使用：类名＋字符串英文名，如 LanguageDialog.remove，这样看到键值就大概知道它表示的字符串是什么含义。不要使用 LanguageDialog.0 这种方式来做键值，如果一定要用，则应该在程序类和资源文件里加上注释。

19.4　plugin.xml 的国际化

plugin.xml 也包含了很多文字信息，例如一个 label 值，它们会显示在界面上，所以也应该国际化，而像 id、class 等就不必国际化了。

plugin.xml 的国际化和类程序的国际化是不同的。本节将把 plugin.xml 中的 actionSet 项的 action 子项国际化，以此演示对 plugin.xml 国际化的方法。

原 plugin.xml 中 action 项的代码如下：

```
<action
        label="样本操作(&amp;S)"
        icon="icons/sample.gif"
        class="cn.com.chengang.SampleAction"
        tooltip="Hello，这是手工做的插件"
        menubarPath="sampleMenu/sampleGroup"
        toolbarPath="sampleGroup"
        id="cn.com.chengang.SampleAction">
</action>
```

19.4.1　创建 plugin.xml 的资源文件

plugin.xml 对应的资源文件名为 plugin.properties、plugin_zh_CH. properties 等，其主名

固定为 plugin，不能改变。

这些文件的位置必须在 myplugin2 或者 myplugin2.language 项目的根目录(和 plugin.xml 同一目录）下。建议将默认的 plugin.properties 放在 myplugin2 项目的根目录下，其他的则放在 myplugin2.language 项目的根目录下。和类程序的资源文件一样，Eclipse 会先找相应语言的资源文件，找不到再去查默认的 plugin.properties。

plugin.xml 资源文件的内容也是“键值 = 翻译字符串”的格式，这里给出默认及中文简体的 properties 文档内容，其他的 properties 以此类推。

文件一：默认的 plugin.properties 如下（本例将默认的设为英文）：

```
actionSets.SampleAction.label=SampleAction (&S)
actionSets.SampleAction.tooltip=Hello,This is my Plugin.
```

文件二：中文简体文档 plugin_zh_CN.properties 的内容如下：

```
#样本操作(&amp;S)
actionSets.SampleAction.label=\u6837\u672C\u64CD\u4F5C(&S)
#Hello，这是手工做的插件
actionSets.SampleAction.tooltip=Hello,\u8FD9\u662F\u624B\u5DE5\u505A\u7684\u63D2\u4EF6
```

说明：

- 键值的取名尽量规范化，最好是像本例一样能从键值看到该项在 plugin.xml 中的位置，以便于今后维护。
- plugin.xml 中的（&S）在 properties 文档中是翻译成（&S），显示为（S）。

19.4.2　修改 plugin.xml 文件

将原文字用键值替换，注意前面还要加一个“%”号，两边的双引号也不能少。

```
<action
      label="%actionSets.SampleAction.label"
      icon="icons/sample.gif"
      class="cn.com.chengang.SampleAction"
      tooltip="%actionSets.SampleAction.tooltip"
      menubarPath="sampleMenu/sampleGroup"
      toolbarPath="sampleGroup"
      id="cn.com.chengang.SampleAction">
</action>
```

19.4.3　运行插件，测试 plugin.xml 的国际化

测试的方法和 19.3 节中测试类程序的国际化是一样的。如果没有设置环境参数-ln，或者设置的语言对应的文档不存在，例如-ln de（德文），则使用默认的 plugin.properties 显示如图 19.11 所示左部的英文。如果设置环境参数为-ln zh_CN，则显示如图 19.11 所示右部的

简体中文。

图 19.11　测试 plugin.xml 的国际化

19.5　其他 XML 文件的国际化

如前面几节创建的几个根目录下的 XML：toc.xml、other_toc.xml、HelpContexts.xml，它们里面都含有中文信息，所以在国际化时也要处理它们。

虽然都是 XML 文件，但这些 XML 文件的国际化和 plugin.xml 是不同的。

19.5.1　在 myplugin2.language 项目创建国际化所需的目录结构

建立如图 19.12 方框中所示的目录结构。注意，其目录名不能任意取名。然后将三个 XML 文件的每个目录复制一份，原 myplugin2 项目的 XML 依然保留作为默认的 XML 文件。

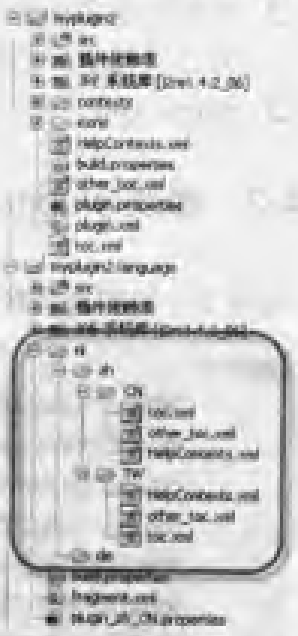


图 19.12　XML 国际化的目录结构

19.5.2　修改各语言的 XML 文档

XML 文档则可以用文本编辑器打开，直接用各种语言文字编辑修改，它不存在资源文件的概念。

19.5.3　运行插件，测试 XML 文件国际化

测试的方法和 19.3 节、19.4 节的都一样。如果没有设置环境参数 -ln，或者设置的语言所对应的 myplugin2.language 项目的目录或文档不存在，例如-ln fr（法文），则会使用 myplugin2 项目下默认的 XML。如果设置环境参数为-ln zh_CN，则使用 myplugin2.language 项目的目录 nl\zh\CN 下的 XML 文件。

19.6　使用“外部化字符串”向导

Eclipse 提供了用于国际化的“外部化字符串”向导，但它只能用于对类程序进行国际

化，而无法用于 plugin.xml 和其他 XML 文件。本节将介绍该向导的使用方法。

19.6.1　运用“外部化字符串”向导的步骤

（1）如图 19.13 所示。在包资源管理器视图中，右键单击 LanguageDialog 文件，然后选择“源代码→外部化字符串”选项。

图 19.13　选择“外部化字符串”选项

（2）在图 19.14 所示的对话框中，将“确定”项的键值改为 LanguageDialog.ok，“删除”项的键值改为 LanguageDialog.remove；将最下方的 properties file 文本框值改为：myplugin2.language/src/cn/com/chengang/messages.properties，表示资源文件将创建在 myplugin2.language 项目的 src/cn/com/chengang 目录下。其他设置都不改动，然后单击“下一步”按钮。由上到下，由左往右说明一下对话框中各项的含义：

- 最顶端的文本框值为 LanguageDialog.，它是资源文件的键值前缀。
- 接下来的列表中，是从 LanguageDialog 代码中提取出来的文字项，显示有“确定”“删除”这两个字串没有国际化。其中 key 列显示的该项国际化后的键值，key 列值的前缀 LanguageDialog 由顶端文本框的值决定。
- 列表右边有三个按钮。选择列表中的文字项后，单击 Externalize 按钮表示要将此项国际化；单击 Ignore 按钮表示跳过，并且以后也不再显示列表中；单击 Internalize 按钮表示跳过，但以后会再次显示在此列表中。单击不同的铵钮，文字项前的图标会相应变化，也可以直接单击该图标来实现这三个按钮的功能。
- 中部的编辑框 Context 显示的是 LanguageDialog 类的源代码。
- 最下部左边的文本框值 cn.com.chengang.Messages，表示负责国际化的 Messages 类将创建在 cn.com.chengang 包中，并取名 Messages。
- 最下部右边的文本框中的值，表示资源文件将创建在 myplugin2.language 项目的 src/cn/com/chengang 目录中，并取名 messages.properties。如果该目录下已经存在 messages.properties 文件，则会将新的内容插入原文件，而不是覆盖；如果该路径下不存在 messages.properties 文件，向导会自动创建一个。

（3）接下来的窗口显示“找到问题”，不过只是提示说“messages.properties 不存在，

本向导将创建它”，不用理会这个提示，直接单击“下一步”按钮。

图 19.14　外部化字符串的主窗口

（4）新出现的窗口里预览了向导将要做的修改，直接单击“完成”按钮。这时向导将做以下三件事：

- 修改 LanguageDialog 类，并在修改类后面会加一个注释//$NON-NLS-1$，它表示下次运行向导时，这行语句的第 1 个字串不会再被向导提示要国际化。
- 在 LanguageDialog 类的同目录下新增了一个 Messages 类：Messages.java。
- 在 myplugin2.language 项目的 cn.com.chengang 包中新建一个资源文件 messages.properties。

19.6.2　向导完成后的处理

向导完成后，资源文件还得再编辑处理。将 messages.properties 文档为英文、简体、繁体等各复制一份，然后分别编辑。Message 类和 myplugin2 项目中的程序则不用再手工编辑了。

19.6.3　总结

“外部化字符串”向导在国际化操作中是最常用的一项功能，它大大简化了软件国际化的工作量，但此向导还不能替代国际化中所有的工作，在向导完成之后，还必须对资源文件进行再处理。

19.7　本 章 小 结

本章介绍了对一个插件项目的类程序、plugin.xml 和其他 XML 文件，进行国际化的完

整方法，以及给出了“外部化字符串”向导的操作流程。

应该注意，项目国际化之后往往会增加系统维护的工作量，所以通常是在整个项目完成之后再开始进行国际化。而且，国际化时的取名一定要规范，尽量以完整的英文来取名，以便今后修改时能顾名知义，利于维护。

第 20 章　报表：用 POI 与 Excel 交互

20.1　POI 概述

20.1.1　POI 简介

如果要开发 Eclipse 插件式的信息应用系统，就肯定会涉及报表输出。本书建议是将报表输出成 Excel，这样用户就可以利用 Excel 的强大功能对报表再编辑润色。

在 Eclipse 中要和 MS Office 交互需要用于一个第三方库：POI。它是 Apache Jakarta 的一个免费的开源项目，可以用于任何 Java 程序中（包括 Web 程序），而不仅仅是 Eclipse。POI 不是 Eclipse 的插件，而是几个 JAR 包，其知识与插件无关，如果读者仅关心插件的开发可以选择跳过此章。

本章示例代码主要借鉴了 POI 包中的相关文档，特此说明。

20.1.2　POI 的下载与安装

1．下载

（1）下载网站：http://jakarta.apache.org/，如图 20.1 所示，单击 Binaries 选项。

（2）如图 20.2 所示，选择 POI 链接。

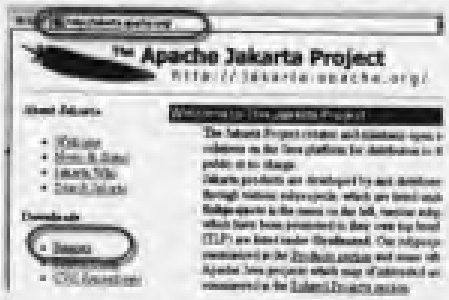


图 20.1　网站主页

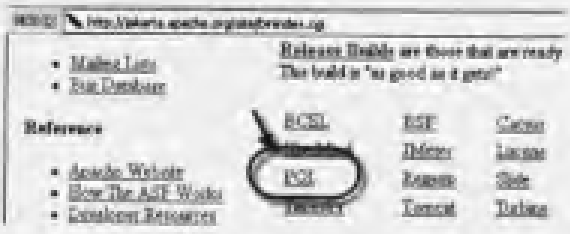


图 20.2　选择下载项目

（3）如图 20.3 所示，右键单击 POI 下的“2.5.1.zip”链接（有可能读者下载时版本已

附录　光盘内容简介

第 1 篇　起步篇（第 1~5 章）

myproject\src\net\yeah\glchengang　第 4 章的 Hellworld。

第 2 篇　SWT_JFace 篇（第 6~16 章）

myswt\src\com　第 6 章实例。

myswt\src\swt　第 8~13 章实例。其中，子目录 thread 为第 13 章的实例。

myswt\src\jface\tableviewer　第 14 章实例。

myswt\src\jface\treeviewer　第 15 章实例。

myswt\src\jface\dialog　第 16 章实例。其中，子目录 wizard 为“向导式对话框”一节的实例；子目录 other 为“其他类型对话框”一节的实例。

第 3 篇　插件开发篇（第 17~22 章）

myplugin　17.2.1 小节“使用向导一步步创建 HelloWorld”的实例。

myplugin2、myplugin2.language　17.2.2 小节及第 18、19 章的实例。

myplugin2\src\poi　第 20 章的实例，它不是插件，而是以应用程序（application）方式运行。

应用程序打包结果　myswt_application 需要 JVM 运行。myswt_application2 可以脱离 JVM 运行。

插件打包结果　“第 21.2 节 插件项目的打包与发行”的结果。

用 Ant 打包的结果　“第 21.3 节　用 Ant 来打包”的结果。

sms　第 22 章插件项目实战的项目文件的 CVS 最终版。

第 4 篇　Web 开发篇（第 23~26 章）

myweb　第 23、24、25、26 章的项目文件的 CVS 最终版。

Cvsfile

第 22~26 章项目的 CVS 备份目录。

视频演示

包含本书涉及的一些软件的下载和安装的视频演示，以及如何安装本光盘的 sms 项目和 myplugin2 项目的视频演示。运行文件是 eclipse.exe。其内容列表如下：

第 1 篇　起步篇（第 1~5 章）

JDK 下载；

JDK 安装；

Eclipse 下载；

Eclipse 多国语言包下载；

Eclipse 和多国语言包的安装；

HelloWorld（4.2 节　创建 Java 项目并运行）；
SWT Designer 下载；
SWT Designer 安装；
SWT Designer 注册方法；
CVSNT 下载；
CVSNT 安装。

第 2 篇　插件开发篇（第 17~22 章）
安装光盘实例（以安装 myplugin2 为例）；
安装光盘的 CVS 备份（以安装 SMS 项目为例）；
MySQL 及其 JDBC 连接包的下载；
MySQL 的安装、SMS 项目数据库的配置。

第 3 篇　Web 开发篇（第 23~26 章）
Tomcat 的安装；
Lomboz 的安装和配置；
Web 应用的 HelloWorld；
不用发布就可以用 IE 显示 JSP 修改效果；
将 JavaBean 重定位到 WEB-INF_classes 目录下。

HelloWorld（4.2 节　创建 Java 项目并运行）；
SWT Designer 下载；
SWT Designer 安装；
SWT Designer 注册方法；
CVSNT 下载；
CVSNT 安装。

第 2 篇　插件开发篇（第 17~22 章）
安装光盘实例（以安装 myplugin2 为例）；
安装光盘的 CVS 备份（以安装 SMS 项目为例）；
MySQL 及其 JDBC 连接包的下载；
MySQL 的安装、SMS 项目数据库的配置。

第 3 篇　Web 开发篇（第 23~26 章）
Tomcat 的安装；
Lomboz 的安装和配置；
Web 应用的 HelloWorld；
不用发布就可以用 IE 显示 JSP 修改效果；
将 JavaBean 重定位到 WEB-INF_classes 目录下。

HelloWorld（4.2 节　创建 Java 项目并运行）；
SWT Designer 下载；
SWT Designer 安装；
SWT Designer 注册方法；
CVSNT 下载；
CVSNT 安装。

第 2 篇　插件开发篇（第 17~22 章）
安装光盘实例（以安装 myplugin2 为例）；
安装光盘的 CVS 备份（以安装 SMS 项目为例）；
MySQL 及其 JDBC 连接包的下载；
MySQL 的安装、SMS 项目数据库的配置。

第 3 篇　Web 开发篇（第 23~26 章）
Tomcat 的安装；
Lomboz 的安装和配置；
Web 应用的 HelloWorld；
不用发布就可以用 IE 显示 JSP 修改效果；
将 JavaBean 重定位到 WEB-INF_classes 目录下。